

A New Fine-Grained Evolutionary Algorithm Based on Cellular Learning Automata

R. Rastegar

Department of Computer Engineering, Amirkabir University of Technology, Tehran, Iran.
ras_reza@yahoo.com

M. R. Meybodi

Department of Computer Engineering, Amirkabir University of Technology, Tehran, Iran.
meybodi@ce.aut.ac.ir

Abstract

In this paper, a new evolutionary computing model, called CLA-EC, is proposed. This model is a combination of a model called cellular learning automata (CLA) and the evolutionary model. In this model, every genome in the population is assigned to one cell of CLA and each cell in CLA is equipped with a set of learning automata. Actions selected by learning automata of a cell determine the genome's string for that cell. Based on a local rule, a reinforcement signal vector is generated and given to the set of learning automata residing in the cell. On the basis of the received signal, each learning automaton in the cell updates its internal structure according to a learning algorithm. The process of action selection and updating the internal structure of learning automata is repeated until a predetermined criterion is met. To show the effectiveness of the proposed model it is used to solve several optimization problems such as real valued function optimization and clustering problems.

Keywords: CLA-EC, Cellular Learning Automata, Evolutionary Algorithm, Learning Automata, Cellular Automata, and Optimization.

1. Introduction

Evolutionary algorithms (EAs) form a class of random search algorithms in which principles of natural evolution are regarded as rules for optimization. They attempt to find the optimal solution to the problem at hand by manipulating a population of candidate solutions. The population is evaluated and the best solutions are selected to reproduce and mate to form the next generation. Over a number of generations, good traits dominate the population, resulting in an increase in the quality of the solutions. They are often applied to optimization problems where specialized techniques such as gradient based algorithms, linear programming, dynamic programming are not available or standard methods fail to give reasonable answers due to multimodality, nondifferentiability or discontinuity of the problem at hand [11]. The basic mechanism in EAs is Darwinian evolution. Bad traits are eliminated from the population because they appear in individuals, which do not survive the process of selection. The good traits survive and are mixed by recombination (crossover) to form better individuals. Mutation also exists in EAs, but it is considered a secondary operator. Its function is to ensure that diversity is not lost in the population, so the EA can continue to explore.

Evolutionary computation can be done in parallel. The basic idea behind most parallel EAs (PEA) is to divide their tasks into chunks and then solve the chunks simultaneously using multiple processors. This divide-and-conquer approach can be applied to EAs in many different ways, and the literature contains many examples of successful parallel implementations. Some parallelization methods use a single population, while others divide the population into several relatively isolated subpopulations. Some methods can exploit massively parallel computer architectures, while others are better suited to multi-computers with fewer and more powerful processing elements. There are four main types of parallel EAs: global single-population master-slave EAs, single-population fine-grained PEA, multiple-population coarse-grained EAs and

hierarchical combinations [12]. In a master-slave EA there is a single panmictic population, but the evaluation of fitness is distributed among several processors. Since in this type of parallel EA, selection and crossover consider the entire population, it is also known as global parallel EAs. Fine-grained parallel PEAs are suited for massively parallel computers and consist of one spatially structured population. Selection and mating are restricted to a small neighborhood, but neighborhoods overlap permitting some interaction among all the individuals. The ideal case is to have only one individual for every processing element available. Multiple-population (or multiple-deme) EAs are more sophisticated, as they consist on several subpopulations that exchange individuals occasionally. This exchange of individuals is called migration and it is controlled by several parameters. More information about fine-grained PEAs can be found in [24], [23], [16], [3], [4], [33], [29] and [1]. Fine-grained PEAs have only one population, but it has a spatial structure that limits the interactions between individuals. An individual can only compete and mate with its neighbors, but since the neighborhoods overlap, good solutions may disseminate across the entire population. It is common to place the individuals of a fine-grained parallel EA in a 1 or 2 or 3-dimensional regular grid. Whitley [33] showed that this scheme could be modeled using Cellular Automata with stochastic rules.

A Cellular Automata (CA) is an abstract model that consists of a large number of simple identical cells with a local interaction. CA is a non-linear dynamical system in which space and time is discrete. It is cellular, because it is made up cells like points in the lattice or like squares of the checker board and it is automata, because it follows a simple rule [34]. It is especially suitable for modeling natural systems that can be described as massive collection of simple objects locally interacting with each other. Informally, a d -dimensional CA consists of a finite or infinite d -dimensional lattice of identical cells. Each cell can assume a state (genome in Fine-grained PEAs) from a finite set of states (search space). The cells update their states synchronously on discrete steps according to a local rule (local selection, crossover and mutation). The new state of each cell depends on previous states of a set of cells, included the cell itself, and constitutes its neighborhood. Decentralization is a feature of CA (and fine-grained PEAs) in which due to large spatial separation of cells (individuals of fine-grained PEA) or limited bandwidth of communication channels, complete information exchange may not be feasible. Each cell (individual) in CA can gather limited information about each other and the overall system (population). Hence, individual cells that have access to partial information must make decision about the changing of cells state (genome). Decentralization, by its nature, introduces uncertainty into the decision. Learning in this decision can overcome the introduced uncertainty.

Learning Automata are general-purpose stochastic optimization tools, which have been developed as models of learning systems [26][31]. They are typically used as the basis of learning systems, which through interactions with a stochastic unknown environment learn the optimal action for that environment. A learning automaton tries to choose, iteratively, the optimal action to apply to environment from a finite number of actions that are available to it. The environment returns a reinforcement signal that shows the relative quality of the action performed by the learning automaton. This signal is given to the learning automaton and the learning automaton adjusts itself using a learning algorithm.

Cellular learning automata (CLA) which is introduced recently is a combination of the cellular automata (CA) and learning automata (LA) [22]. This model is superior to CA because of its ability to learn and also is superior to a single learning automaton because it is a collection of learning automata, which can interact with each other and solve a particular problem. The basic idea of CLA, which is a super class of stochastic CA, is to use learning automata to adjust the state transition probability of stochastic CA [6]. So far, CLA has been used in many applications including image processing [17][21], VLSI placement [19], rumor diffusion [20], commerce networks [18], channel assignment in cellular mobile systems [7], call admission control in cellular mobile system [5], cooperation in multiagent systems [15], load balancing in computational grid [10]. For more information about CLA the reader may refer to [6].

In this paper, using CLA, a new fine-grained PEA called cellular learning automata based evolutionary computing (CLA-EC) is proposed. CLA-EC is obtained by combining cellular learning automata (CLA) model and the evolutionary computing model. In CLA-EC, each genome in the population is assigned to one cell of CLA and each cell in CLA is equipped with a set of learning automata. The set of actions selected by the set of learning automata determines the string genome for that cell. Based on a local rule, a reinforcement signal vector is generated and given to the set of learning automata residing in the cell. On the basis of the received signal, each learning automaton in a cell updates its internal structure according to a learning

algorithm. The process of action selection and updating the internal structure of learning automata is repeated until a predetermined criterion is met.

CLA-EC proposed in this paper is different from the fine-grained PEAs reported in the literatures in two respects: the definition of the local rule and the structure of the individuals. In fine-grained PEA, local rule is a combination of local selection, crossover and mutation, but in CLA-EC, the local rule is a combination of local selection, and the reinforcement signal vector generation schema. The individual in fine-grained PEAs is commonly a string genome, whereas in CLA-EC an individual is composed of a string genome and a model genome (a set of LA).

CLA-EC is not the first evolutionary algorithm in which LA is used. In [25] an algorithm called StGA considers the whole population to be a single learning automaton. Each genome in population is considered as an action of an automaton and as such the fitness of each is replaced by a probability. The evaluation of a genome in the environment results in updating of all genomes probabilities. When the crossover operator is applied, the offspring's fitness is set to the average of the two parents. This algorithm is limited to small populations of short genome length. In [14] an algorithm called GLA is proposed. In this algorithm, a corresponding population of probability genomes replaces the population of bit string genomes in genetic algorithm. The value at each position of each member of the population defines the probability of the allele value in a corresponding bit string of being one at that position. A population of bit strings is generated stochastically at each generation by sampling the probability distributions of the population. A single bit string is created every generation by each of the probability strings. Each bit string then has its performance evaluated in the test environment in a similar manner to that of the GA. The fitness of probability string being equal to that of corresponding bit string. The probabilities at each position are regarded as action selection probabilities of a two-action learning automaton. The probabilities are updated at each generation on basis of a learning algorithm. Mutation operator is not defined and crossover operator is defined as crossover in GA. In [28] an estimation of distribution algorithm called LAEDA is presented. In LAEDA, a set of learning automata is used to model the population of genomes. At each generation a population of genome is generated by the set of learning automata. Then a predefined number of genomes are selected from this population according to a selection strategy. Based on the selected genomes a reinforcement signal vector is generated and given to the set of learning automata. The set of learning automata uses the reinforcement signal vector and update their internal structures. The above process is iterated until a predefined termination condition is met. In this algorithm mutation or crossover operators are not defined.

The CLA-EC differs from StGA and GLA models in two respects. 1: Unlike StGA and GLA, CLA-EC model does not use crossover and mutation operators. 2: Unlike StGA and GLA, in CLA-EC each genome interacts only with its predefined neighboring genomes. The CLA-EC is similar to LAEDA because both don't have crossover and mutation operators. A CLA-EC uses different sets of learning automata to model different genome whereas LAEDA uses one set of learning automata to model the whole population. The CLA-EC and LAEDA both use a subset of selected genomes from the population to generate the reinforcement signal. For both LAEDA and CLA-EC crossover and mutation operators are not defined.

The rest of this paper is organized as follows. Section 2 briefly describes learning automata and cellular learning automata. Section 3 introduces the proposed CLA-EC. The experimental results are given in Section 4. Section 5 is the conclusion.

2- Cellular Learning Automata

In this section we briefly review learning automata, and cellular learning automata.

Learning Automata: Learning Automata are adaptive decision-making devices that operate on unknown random environments. A learning Automaton has a finite set of actions and each action has a certain probability (unknown for the automaton) for getting rewarded by its environment. The aim is to learn to choose the optimal action (i.e. the action with the highest probability of being rewarded) through repeated interaction with 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.

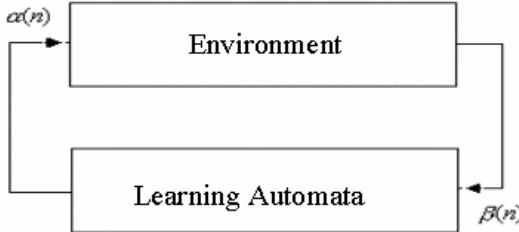


Figure 1: The interaction between learning automata and environment

Figure 1 illustrates how a stochastic automaton works in a 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) [26][31]. A VSLA is a quintuple $\langle \alpha, \beta, p, T(\alpha, \beta, p) \rangle$, where α, β, p are an action set of s actions, an environment response set and the probability set p containing s probabilities, each being the probability of performing each action in the current internal automaton state, respectively. 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 $\beta = \{0, 1\}$. Let $n \in N$ be the set of nonnegative integers. A general linear schema for updating action probabilities can be represented as follows. Let action i be performed.

If $\beta(n)=0$ (reward),

$$p_i(n+1) = p_i(n) + a[1 - p_i(n)] \\ p_{j \neq i}(n+1) = (1 - a)p_j(n)$$

If $\beta(n)=1$ (penalty),

$$p_i(n+1) = (1 - b)p_i(n) \\ p_{j \neq i}(n+1) = (b/s - 1) + (1 - b)p_j(n)$$

where a and b are reward and penalty parameters. When $a=b$, it is called L_{RP} . If $b=0$ it is called L_{RI} and if $0 < b < a < 1$, it is called L_{ReP} . Figure 2 shows the working mechanism of VSLA.

```

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 action and return a reinforcement signal β
  Update probability vector using learning rule
End While

```

Figure 2. Pseudocode of variable-structure learning automaton

Cellular Learning Automata: The potential of learning automaton can be realized completely when multiple automata interact with each other. Interaction may assume different forms such as tree, mesh, or array. Depending on the problem that needs to be solved, one of these structures may be used. In most applications, full interaction between all learning automata is not necessary and is not natural. In the other hand, cellular automaton is a mathematical model for systems consisting of a large number of simple identical components with local interactions. Cellular Learning Automata (CLA), which is obtained by combining cellular automata and learning automata, like cellular automata consists of a large number of simple components each of which have learning capability and act together to solve a particular problem[22]. A CLA is a cellular automata in which each cell is equipped with a learning automaton. The learning automaton residing in a cell uses its action probability and determines the state (action) of that cell. Like CA, there is a rule that CLA operates under it. The rule of CLA and the actions selected by the neighboring learning automata determine the reinforcement signal to the learning automata residing in a cell.

The operation of cellular learning automata can be described as follows: At each step, the learning automaton, residing in a cell chooses an action on the basis of its action probability vector. Initially, an automaton chooses its action either on the basis of past experience or at random. Then, the rule of cellular

automata determines the reinforcement signal to the learning automaton residing in that cell. Finally, the learning automaton updates its action probability vector on the basis of supplied reinforcement signal, the chosen action, and the learning algorithm. The process of action selection and probability vector updating is continued until the desired result is obtained. The CLA is synchronous if all cells are synchronized with a global clock and activated at the same time and it is uniform if each cell uses the same local rule. In this paper we use uniform asynchronous CLA [6].

3. Cellular Learning Algorithm Based Evolutionary Computing

Before proposing the new algorithm, some properties of human society (Table. 1), which approach us to the proposed algorithm, are discussed. Each human society has a number of individuals as its basic units. Each individual makes certain choices, adopts certain attitudes, and operate in certain emotional modes. Individuals affect each other mutually and locally within a certain neighborhood and the information about them is local as well. Learning and adaptation is also an important feature of individuals in a human society. Each individual tries to learn from his own discoveries and previous experiences and the experiences of the neighboring individuals. To learn from experiences, it is a simple strategy in that the individual selects a set of neighboring individuals that are the best with respect to some criteria and then votes between their experiences. Then the individual can update his belief according to the voting results and previous self-experience.

Table 1, Similarities between CLA-EC and Human Society

Basic Units	Cells are the basic units or atoms of CLA-EC.	Individuals are basic units of a society
Possible State	The cells are in states taken from possible solutions.	Individuals make certain choices, adopt certain attitudes, and operate in certain emotional modes.
Interdependence	The state of a cell affects the state of its neighbors and vice versa.	Individuals affect each other mutually.
Locality	Rules are local.	Individuals affect each other only locally, i.e. within a certain neighborhood, and the information about them is local as well.
Overlapping	Neighborhoods are overlapping.	Often interactions have an overlapping structure.

The above discussion motivated us to introduce CLA-EC. In CLA-EC, similar to other evolutionary algorithms, the parameters of the search space are encoded in the form of genomes. Each genome has two components, model genome and string genome. Model genome is a set of learning automata. The set of actions selected by this set of learning automata determines the second component of the genome. Using a local rule, a reinforcement signal vector is generated for each cell and given to the set of learning automata residing in that cell. On the basis of the received signal, each learning automaton updates its internal structure according to a learning algorithm. Then, each cell in CLA-EC generates a new string genome and compares its fitness with the fitness of its own string genome. If the fitness of the generated genome is better than the fitness of the string genome of the cell, the generated string genome becomes the string genome of that cell. The process of generating string genome by the cells is repeated until a termination condition is satisfied. The main issues involved in designing a CLA-EC to solve a given problem are defining a suitable genome representation, designing a fitness function, and the determination of the parameters of CLA (such as the number of cells (population size), topology of CLA and the type of learning automata for each cell).

In the rest of this section, we show how CLA-EC can be used for function optimization problems. Assuming a binary finite search space, a function optimization problem can be presented as the minimization of real function $f: \{0,1\}^m \rightarrow \mathcal{R}$, that is to find

$$\min \{f(\xi) | \xi \in \{0,1\}^m\}.$$

In order to use CLA-EC to optimize function f , first a set of learning automata is assigned to each cell of CLA-EC. The number of learning automata assigned to a cell is the number of bits in the string genome

representing points in the search space of f . Each automaton has two actions called action 0 and 1 and the probability vectors of all learning automata in all cells are initialized to $(0.5, 0.5)$. Then the following steps will be repeated until a termination criterion is met.

1-Every automaton in cell i chooses one of its actions using its action probability vector.

2-Every Cell i generates a new string genome, new^i . The new generated string genome is obtained by concatenating the actions of the automata (0 or 1) residing in that cell.

3- Every cell i computes the fitness value of string genome new^i ; if the fitness of this string genome is better than the one in the cell then the new string genome new^i becomes the string genome of that cell. That is

$$\xi_{n+1}^i = \begin{cases} \xi_n^i & f(\xi_n^i) \leq f(new_{n+1}^i) \\ new_{n+1}^i & f(\xi_n^i) > f(new_{n+1}^i) \end{cases}$$

4- S_e cells of the neighboring cells of cell i are selected. This Selection is based on the fitness value of the neighboring cells according to a selection strategy such as truncation strategy.

5- Every cell i on the basis of the selected cells, generates a reinforcement vector and uses it as the input to the set of learning automata associated to the cell. Let $N_s(i)$ be the set of selected neighbors of cell i . Define,

$$N_{i,j}(k) = \sum_{l \in N_s(i)} \delta(\xi_n^{l,j} = k),$$

where,

$$\delta(\text{exp}) = \begin{cases} 1 & \text{if exp is true} \\ 0 & \text{otherwise} \end{cases}$$

$\beta_n^{i,j}$, the reinforcement signal given to learning automaton j of cell i , is computed as follows,

$$\beta_n^{i,j} = \begin{cases} u(N_{i,j}(1) - N_{i,j}(0)) & \text{if } \xi_n^{i,j} = 0 \\ u(N_{i,j}(0) - N_{i,j}(1)) & \text{if } \xi_n^{i,j} = 1 \end{cases},$$

where $u(\cdot)$ is a step function. The overall operation of CLA-EC is summarized in the algorithm of figure 3.

```

Initialize.
While not done do
  For each cell  $i$  in CLA do in parallel
    Generate a new string genome
    Evaluate the new string genome
    If  $f(\text{new string genome}) > f(\text{old string genome})$  then
      Accept the new string genome
    End if
    Select  $S_e$  cells from neighbors of cell  $i$ 
    Generate the reinforcement signal vector
    Update LAs of cell  $i$ 
  End parallel for
End while

```

Figure 3. Pseudo code of CLA-EC

Remark. The CLA-EC has a very high degree of diversity in the initial population and also during the early generations. This property decreases the probability of premature convergence. The CLA-EC also works over a population that is distributed over the search space and therefore making it less susceptible to become trapped in a local optima.

4. Simulation results

In this section we study the performance of CLA-EC by conducting extensive simulations on two classes of optimization problems: function optimization and data clustering problems. The CLA-EC used for these

simulations has a linear topology with wrap around connection and q cells. If the radius of neighborhood is one the neighbors of cell i are cell $i-1$ and cell $i+1$ as indicated in figure 4.

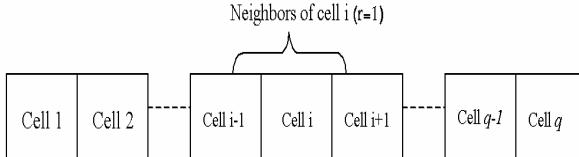


Figure 4. A one-Dimensional (linear) CLA-EC with neighborhood radius one ($r=1$) and wrap around connection and q cells.

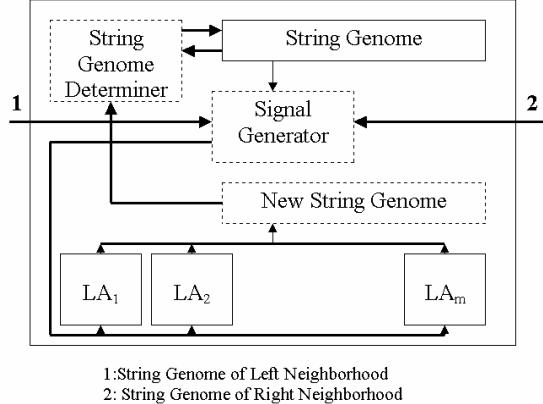


Figure 5: The structure of a cell

The architecture of a cell is shown in figure 5. Each cell is equipped with m learning automata. The string genome determiner compares the new string genome with the string genome residing in the cell. The string with the higher quality replaces the string genome of the cell. Depending on the neighboring string genomes and the string genome of the cell, a reinforcement signal will be generated by the signal generator. In this paper we study the performance of CLA-EC with linear topology and neighborhood radii 1 and 2.

4.1. Function optimization

This section presents simulation results for five function optimization problems and then compare these results with results obtained using Simple Genetic Algorithm (SGA), Population based Incremental Learning (PBIL) and Compact Genetic Algorithm (cGA) in terms of solution quality, and the number of function evaluations taken by the algorithm to converge completely for a given population size. A CLA-EC completely converges when all learning automata residing in all cells converge, i.e. the population (the set of string genomes residing in cells of CLA-EC) remains unchanged. Each quantity of the reported results is the average taken over 20 runs. The population size (number of cells in CLA-EC) varies from 3 to 49 with increments of two. The proposed Algorithm is tested for L_{RI} and L_{RP} learning algorithms. For the sake of convenience in presentation, we use $CLA-EC(automata(a,b),r,se,q)$ to refer to CLA-EC algorithm with q cells, neighborhood radius r , the number of selected cell Se when using learning automaton *automata* with reward parameter a and penalty parameter b . The algorithm terminates when all learning automata converge. The proposed algorithm is tested on five different standard function minimization problems. These functions that are given below are borrowed from reference [8].

1) First De Jong function (sphere)

$$F_1(X) = \sum_{i=1}^3 x_i^2 \quad -5.12 \leq x_i \leq 5.12$$

$F_1(X)$ is considered to be a very simple task for every serious minimization method. The minimum is 0.

2) Second De Jong function (Rosenbrock's saddle)

$$F_2(X) = 100(x_1^2 - x_2)^2 + (1 - x_1)^2 \quad -2.048 \leq x_i \leq 2.048$$

Although $F_2(X)$ has just two parameters, it has the reputation of being a difficult minimization problem. The minimum is 0.

3) Third De Jong function (step)

$$F_3(X) = \sum_{i=1}^5 \text{integer}(x_i) \quad -7.12 \leq x_i \leq 7.12$$

The step function exhibits many plateaus, which pose a considerable problem for many minimization algorithms.

4) Fourth De Jong function (quartic)

$$F_4(X) = \sum_{i=1}^{30} ix_i^4 + \text{Gauss}(0,1) \quad -1.28 \leq x_i \leq 1.28$$

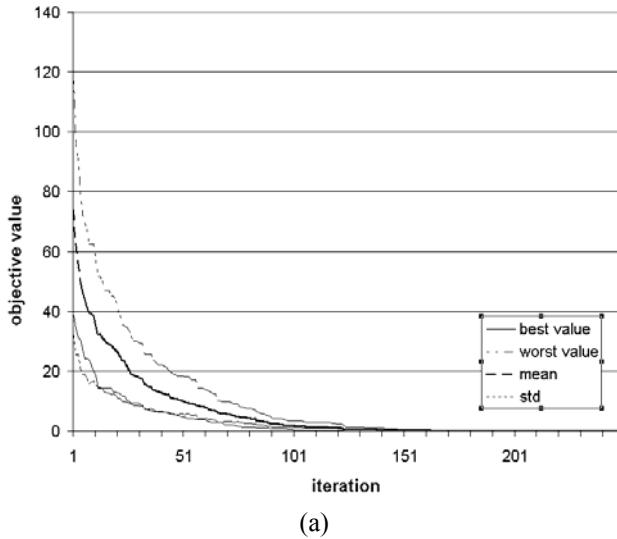
This function is designed to test the behavior of a minimization algorithm in the presence of noise.

5) Fifth De Jong function (Shekel's Foxholes)

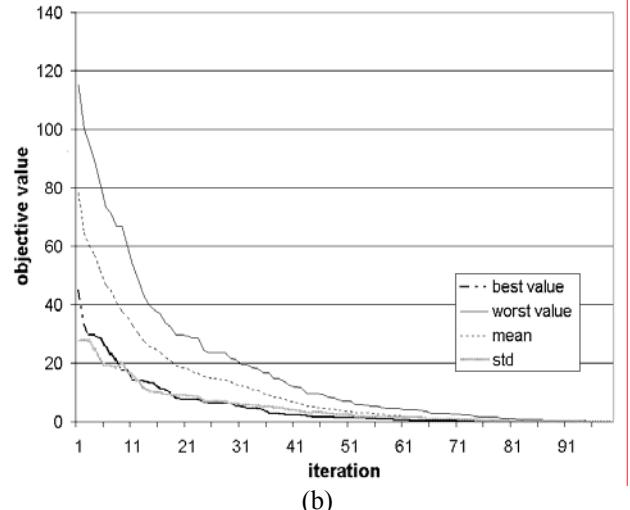
$$F_5(X) = (0.002 + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^2 (x_i - a_{ij})^6})^{-1} \quad -65.536 \leq x_i \leq 65.536$$

To study the speed of the convergence of CLA-EC, we report the best, the worst, the mean and the standard deviation of fitnesses of all cells for each of the function optimization problem. Figure 6 to 10 show the result of simulations for CLA-EC($L_{RP}(0.01,0.01),1,2,5$) and CLA-EC($L_{RI}(0.01,0.01),1,2,5$). Simulation results indicate that CLA-EC converges to a near optimal solution.

The size of CLA-EC (population size) is another important parameter, which affects the performance of CLA-EC. Figures 11 through 14 show the effect of the size of CLA-EC on speed of convergence of CLA-EC($L_{RP}(0.01,0.01),1,2, -$) and CLA-EC($L_{RI}(0.01,0.01),1,2, -$). Each point in these figures shows the best fitness obtained at one iteration. Simulation results show that as the size of CLA-EC increases the speed of convergence increases. However, it is observed that for some experiments, there exists a value if the size of the CLA-EC increases beyond that, no increase in the performance occurs [27].

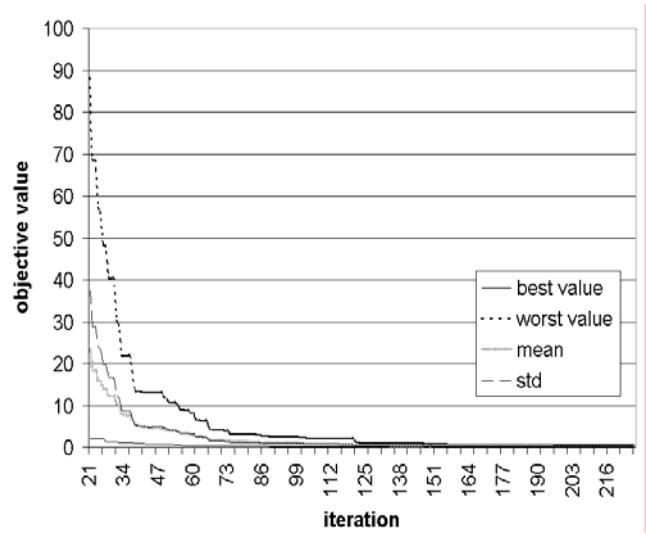
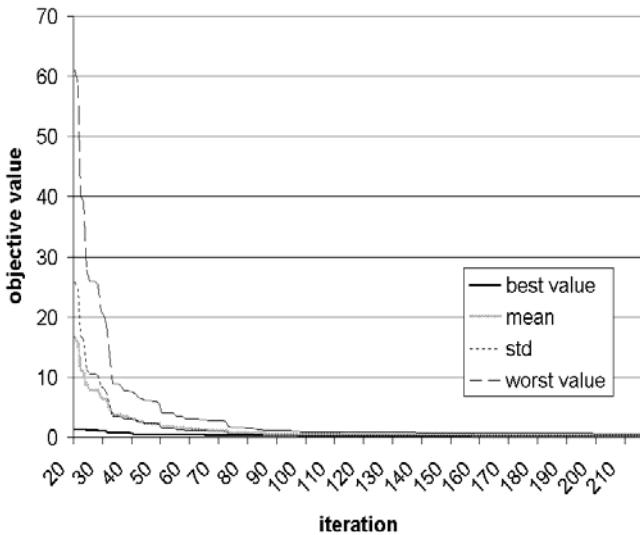


(a)



(b)

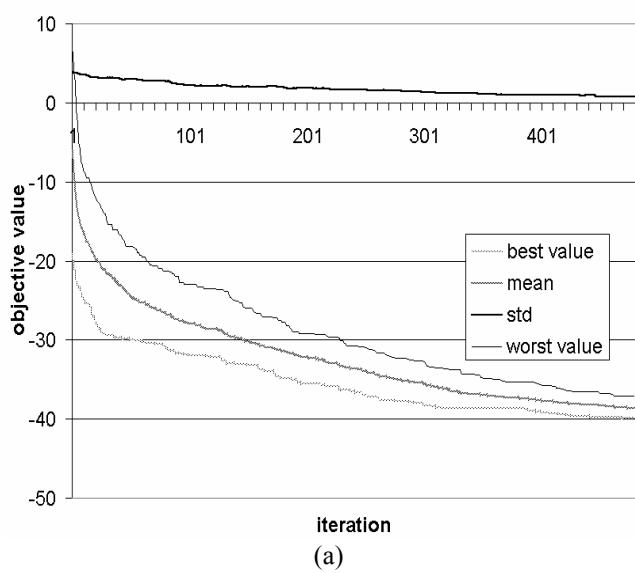
Figure 6: function F_1
(a) CLA-EC($L_{RP}(0.01,0.01),1,2,5$) **(b) CLA-EC($L_{RI}(0.01,0.01),1,2,5$)**



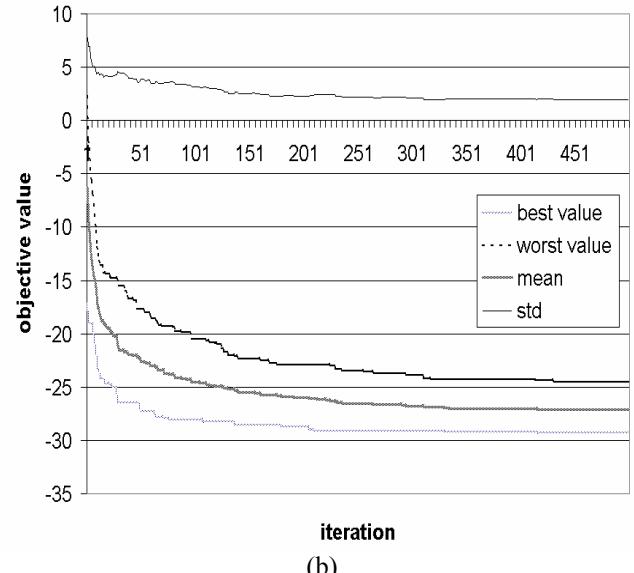
(a)

(b)

Figure 7: function F_2
(a) CLA-EC($L_{RP}(0.01,0.01),1,2,5$) **(b) CLA-EC($L_{RI}(0.01,0.01),1,2,5$)**

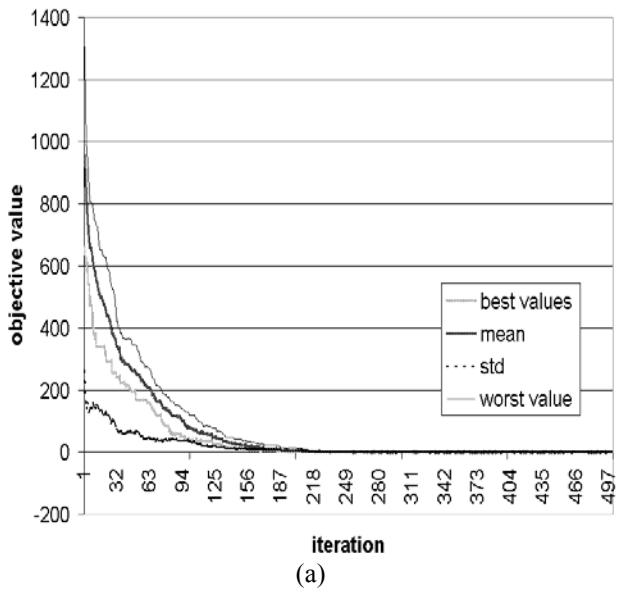


(a)

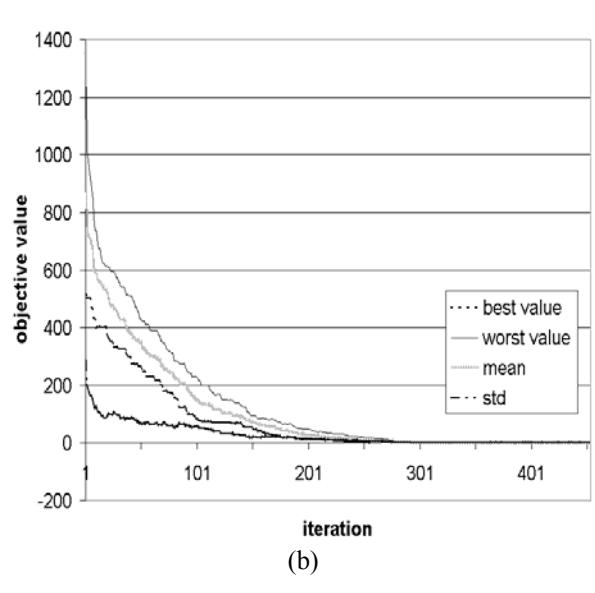


(b)

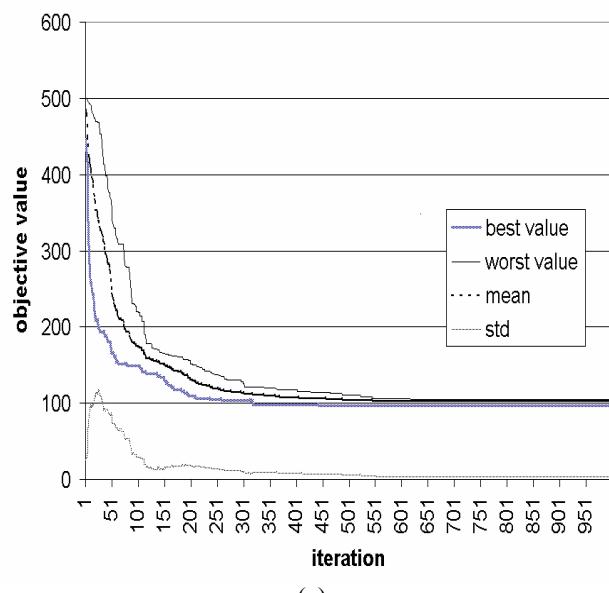
Figure 8: function F_3
(a) CLA-EC($L_{RP}(0.01,0.01),1,2,5$) **(b) CLA-EC($L_{RI}(0.01,0.01),1,2,5$)**



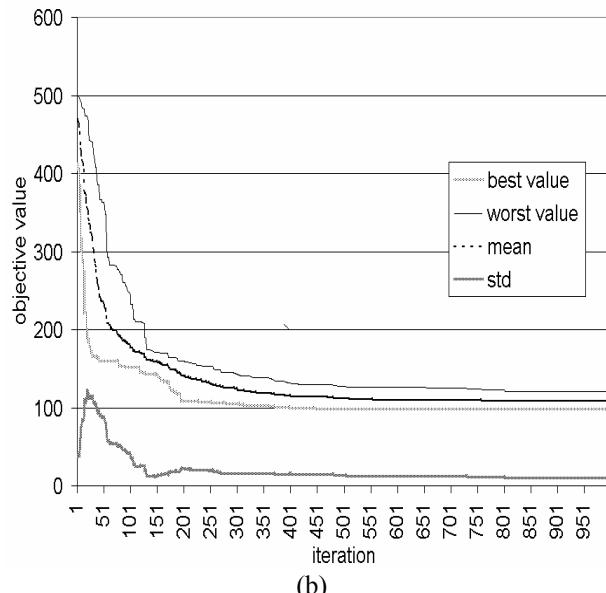
(a)



(b)

Figure 9: function F_4 (a) CLA-EC($L_{RP}(0.01,0.01)$,1,2,5)(b) CLA-EC($L_{RI}(0.01,0.01)$,1,2,5)

(a)



(b)

Figure 10: function F_5 (a) CLA-EC($L_{RP}(0.01,0.01)$,1,2,5)(b) CLA-EC($L_{RI}(0.01,0.01)$,1,2,5)

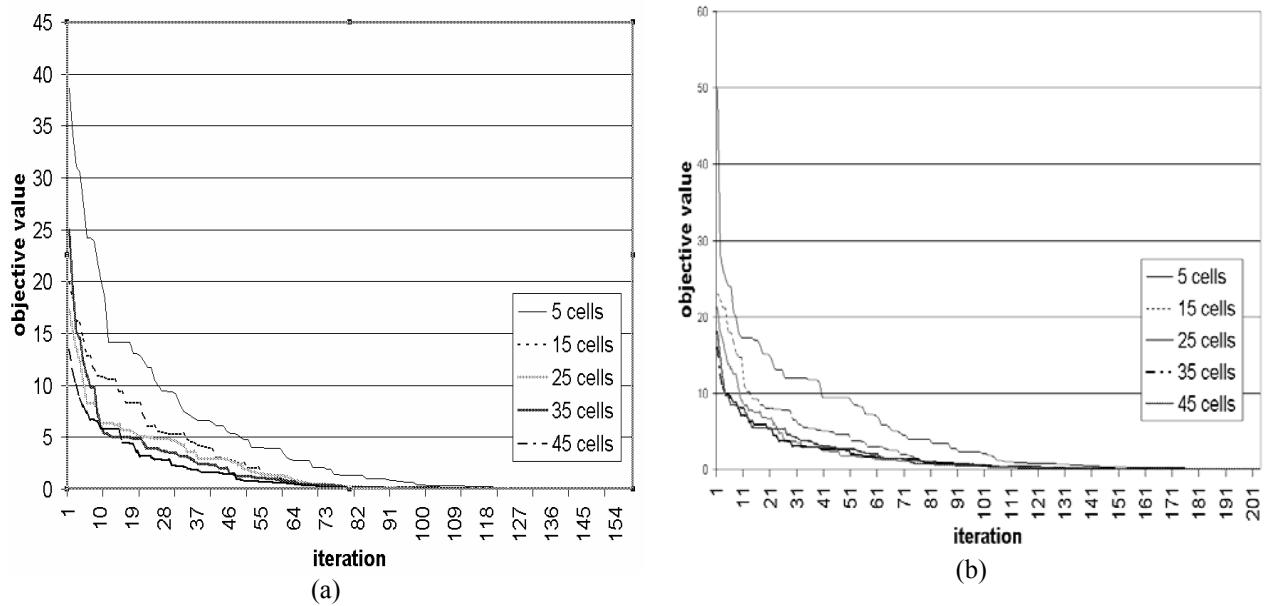


Figure 11: the best value obtained at each iteration for function F_1
(a) CLA-EC($L_{RP}(0.01,0.01)$, 1, 2, -) (b) CLA-EC($L_{RI}(0.01,0.01)$, 1, 2, -)

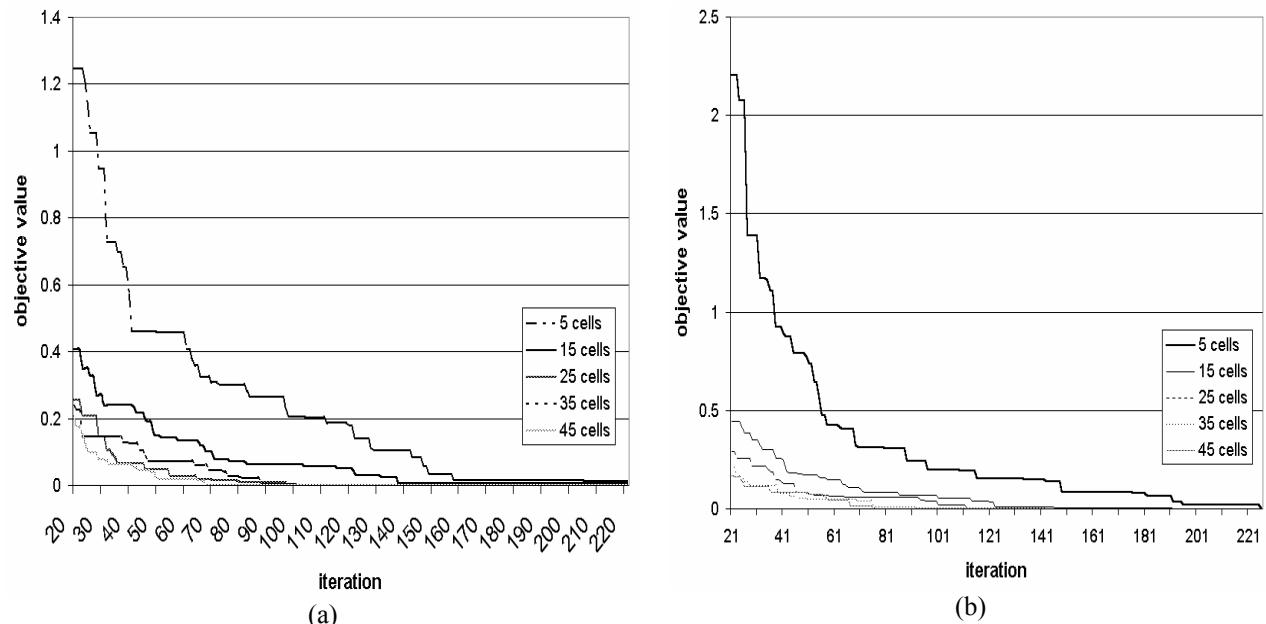


Figure 12: the best value obtained at each iteration for function F_2
(a) CLA-EC($L_{RP}(0.01,0.01)$, 1, 2, -) (b) CLA-EC($L_{RI}(0.01,0.01)$, 1, 2, -)

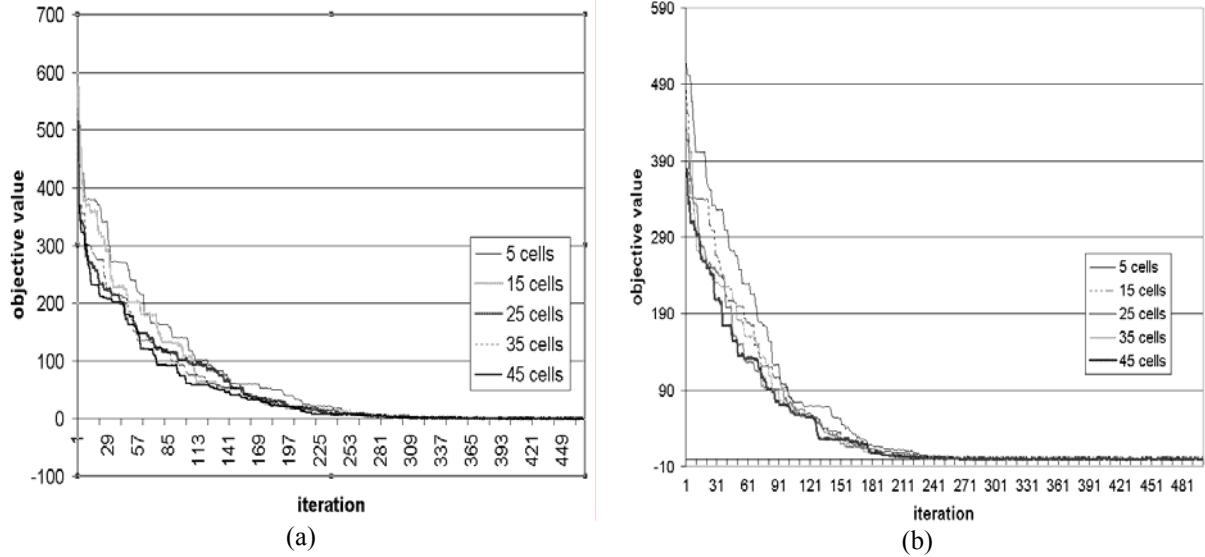


Figure 13: the best value obtained at each iteration for function F_4
(a) CLA-EC($L_{RP}(0.01,0.01)$,1,2, -) **(b) CLA-EC($L_{RI}(0.01,0.01)$,1,2, -)**

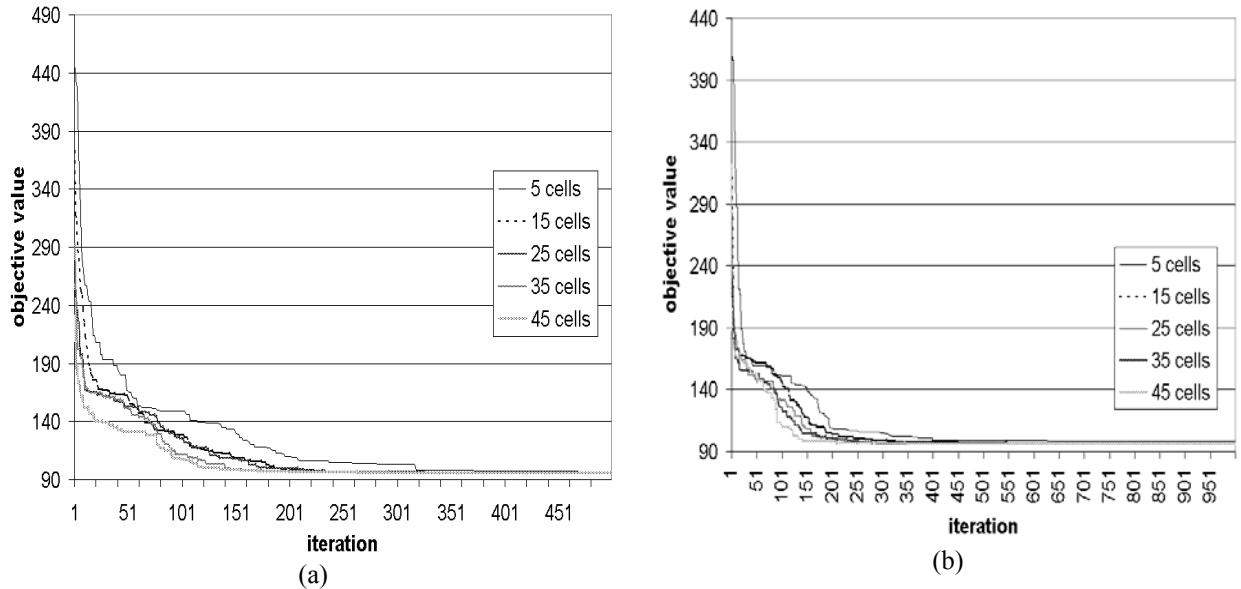


Figure 14: the best value obtained at each iteration for function F_5
(a) CLA-EC($L_{RP}(0.01,0.01)$,1,2, -) **(b) CLA-EC($L_{RI}(0.01,0.01)$,1,2, -)**

To study the effect of neighborhood radius and Se on the performance of CLA-EC, several simulations are conducted and results are reported in figures 15 through 24. Neighborhood radii one and two are considered. Most of simulations show that CLA-EC with radius one produces high quality solutions. For F_1 , F_2 , F_4 when Se is set to 2 or 4, CLA-EC performs better in terms of speed of convergence and the quality of solutions obtained. For F_3 , when Se is set to 2 or 4, CLA-EC converges faster but solutions obtained are not optimal. When Se is set to 1, 3, or 5, CLA-EC finds the optimal solution in all runs. For F_5 when Se is set to 3 or 5, the rate of convergence of e CLA-EC increases.

Remark. For the neighborhood radius of r each cell has $2r+1$ neighbors; in simulations when r is 1 (2) then Se is considered to be 1, 2 or 3 (1, 2, 3,4 or 5). Choosing $2r+1$ as the value of Se simplifies step 4 of the CLA-EC algorithm. This is because the algorithm no longer requires any selection strategy and hence lower time complexity for the algorithm.

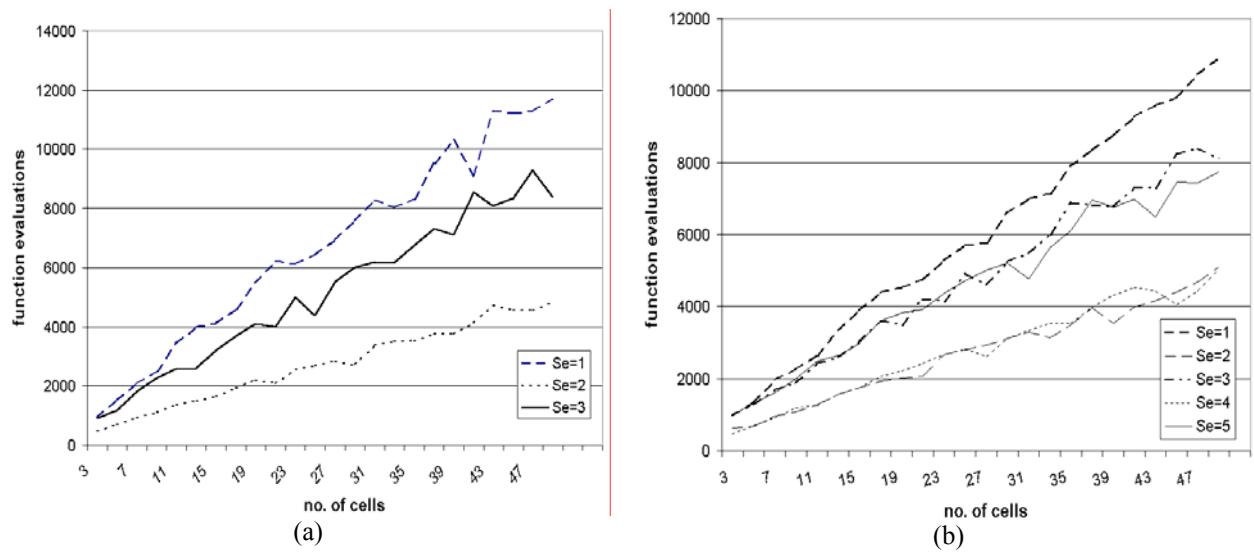


Figure 15: function F_1
a) CLA-EC($\text{LRP}(0.01, 0.01)$, 1, -, -) **b) CLA-EC($\text{LRP}(0.01, 0.01)$, 2, -, -)**

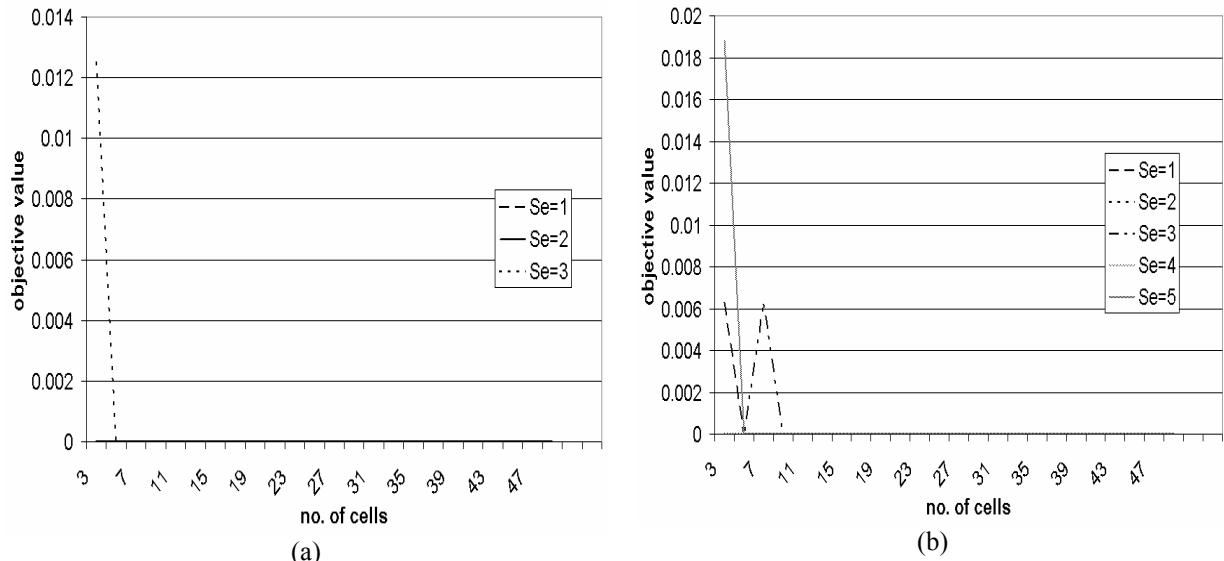


Figure 16: function F_1
a) CLA-EC($\text{LRP}(0.01, 0.01)$, 1, -, -) **b) CLA-EC($\text{LRP}(0.01, 0.01)$, 2, -, -)**

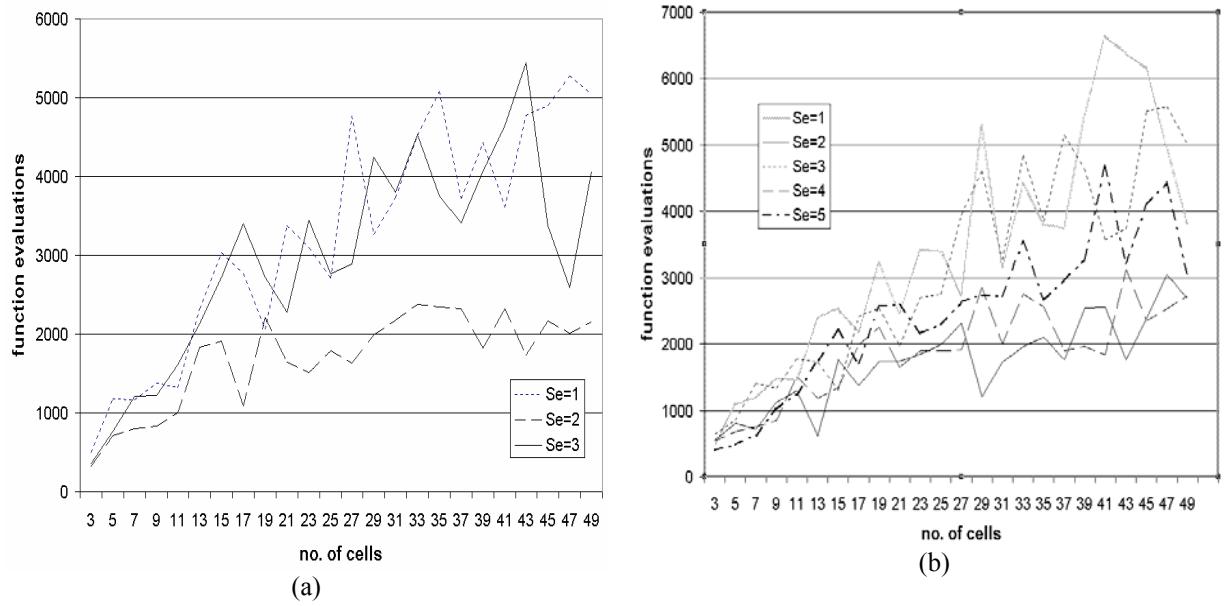


Figure 17: function F_2
a) CLA-EC($LRP(0.01,0.01)$,1,-,-) **b) CLA-EC($LRP(0.01,0.01)$,2,-,-)**

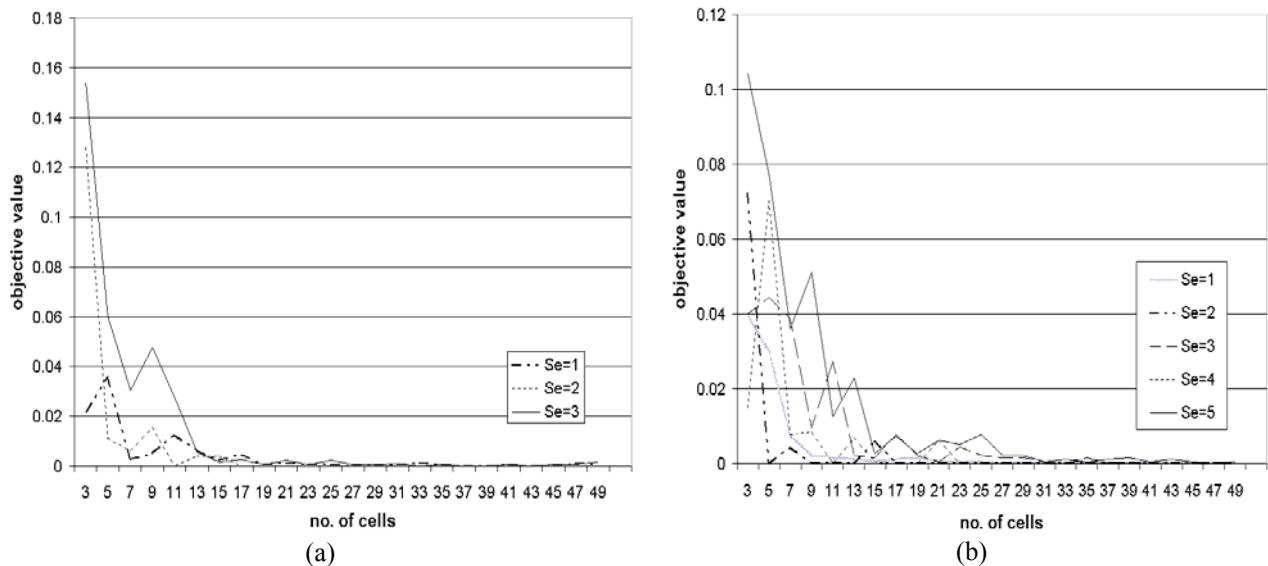
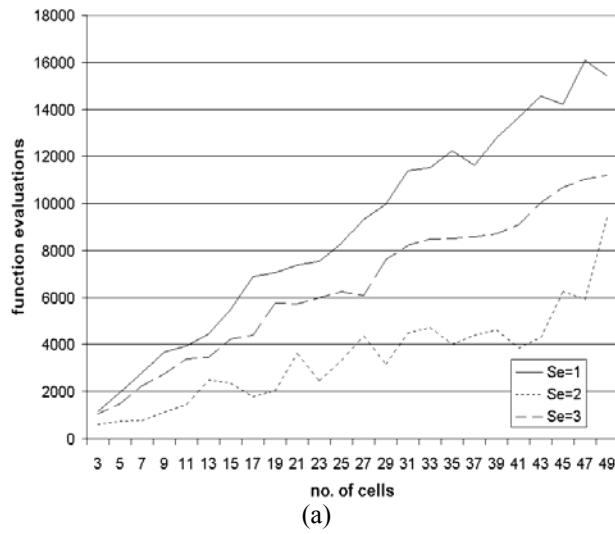
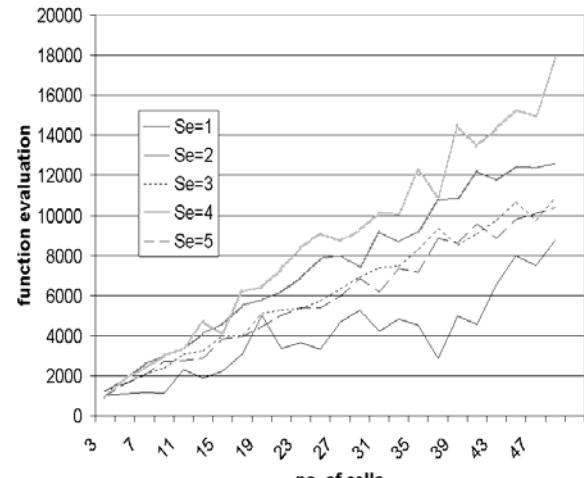


Figure 18: function F_2
a) CLA-EC($LRP(0.01,0.01)$,1,-,-) **b) CLA-EC($LRP(0.01,0.01)$,2,-,-)**



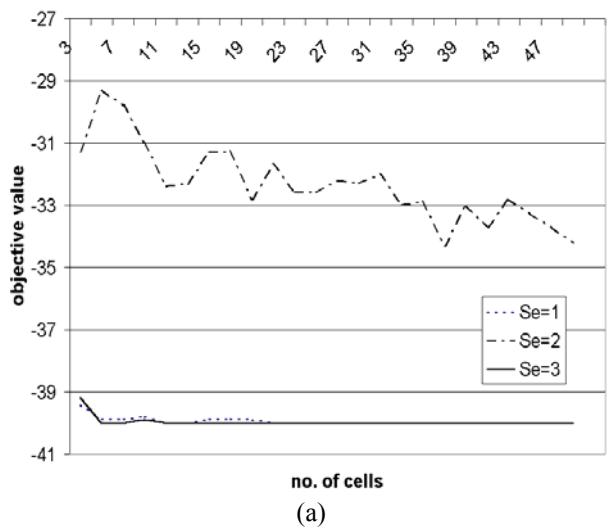
(a)



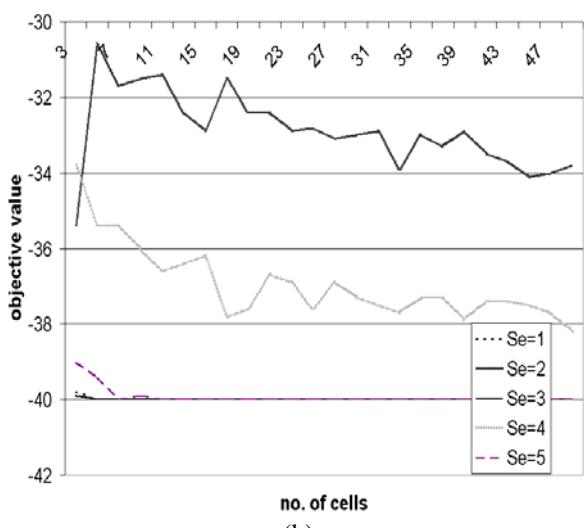
(b)

Figure 19: function F_3

a) CLA-EC(LRP(0.01,0.01),1, - , -) b) CLA-EC(LRP(0.01,0.01),2, - , -)



(a)



(b)

Figure 20: function F_3

a) CLA-EC(LRP(0.01,0.01),1, - , -) b) CLA-EC(LRP(0.01,0.01),2, - , -)

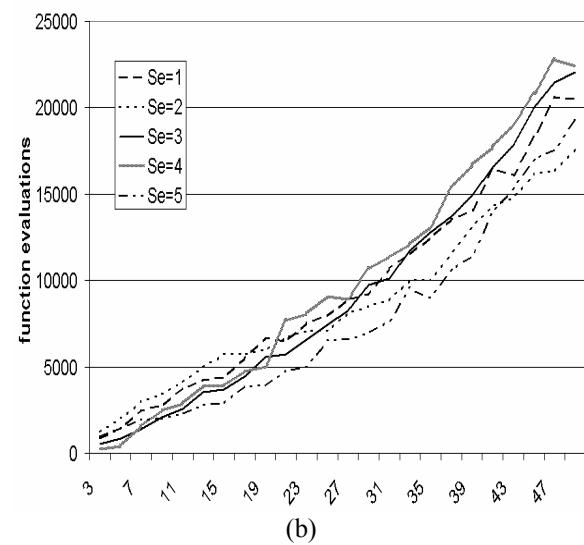
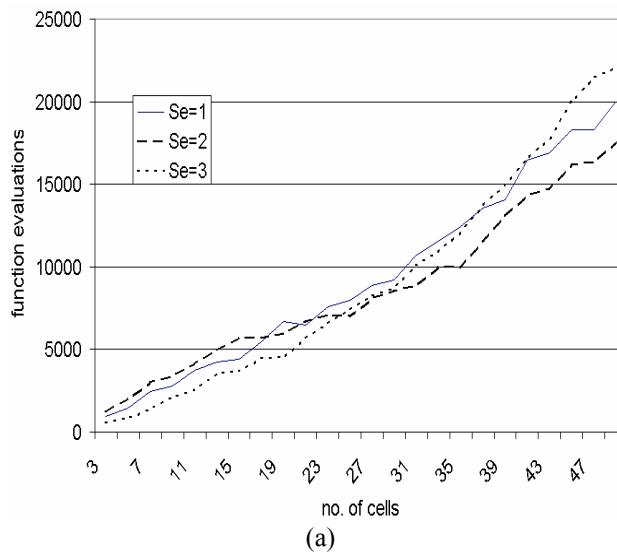


Figure 21: function F_4
a) CLA-EC(LRP(0.01,0.01),1,-,-) **b) CLA-EC(LRP(0.01,0.01),2,-,-)**

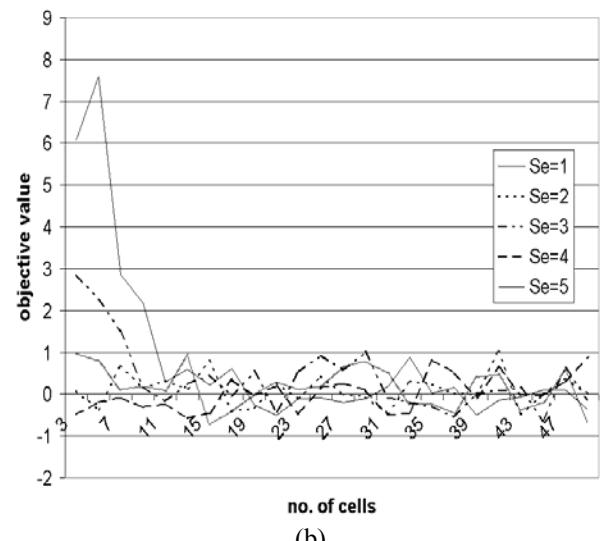
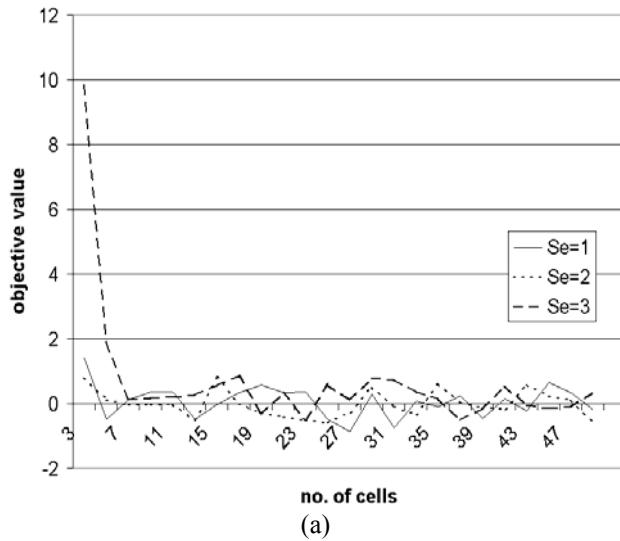


Figure 22: function F_4
a) CLA-EC(LRP(0.01,0.01),1,-,5) **b) CLA-EC(LRP(0.01,0.01),2,-,5)**

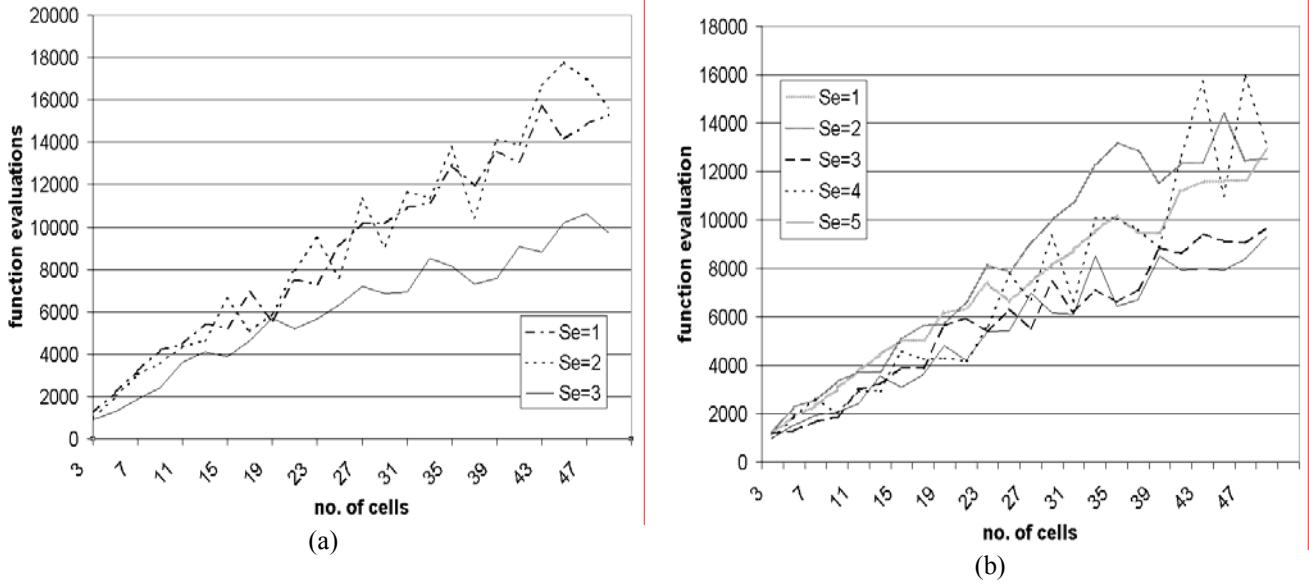


Figure 23: function F_5
a) CLA-EC($LRP(0.01,0.01)$,1,-,5) b) CLA-EC($LRP(0.01,0.01)$,2,-,5)

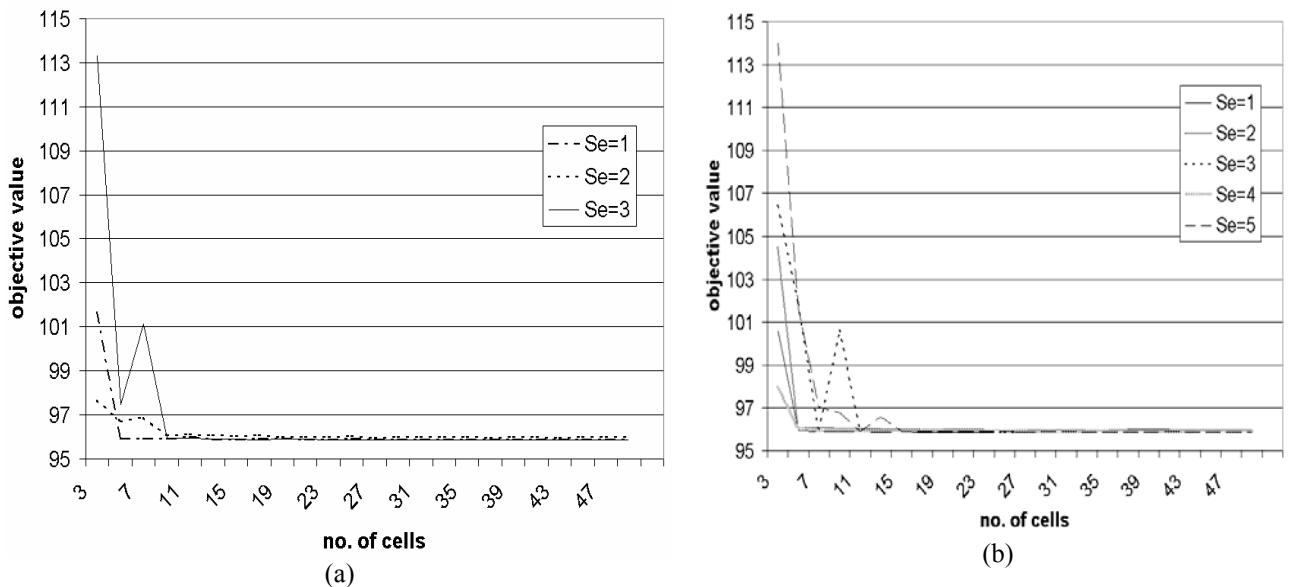


Figure 24: function F_5
a) CLA-EC($LRP(0.01,0.01)$,1,-,5) b) CLA-EC($LRP(0.01,0.01)$,2,-,5)

Figure 25 shows the evolution of string genomes of $CLA-EC(L_{RP}(0.01,0.01),1,2,10)$ for function F_2 during the optimization process. At the beginning of the algorithm, the genome diversity of $CLA-EC$ is high and decreases as $CLA-EC$ approaches the solution. After the $CLA-EC$ converges, all the genomes in the $CLA-EC$ will be in the convergence area as shown in figure 25. That is, all string genomes are either optimal or near the optimal solution of F_2 . Figure 26 shows the evolution of the fitness of all genomes of $CLA-EC(L_{RP}(0.01,0.01),1,2,10)$ during a typical run. This figure indicates that the fitnesses of genomes in $CLA-EC$ gets closer to each other as the optimization process proceeds.

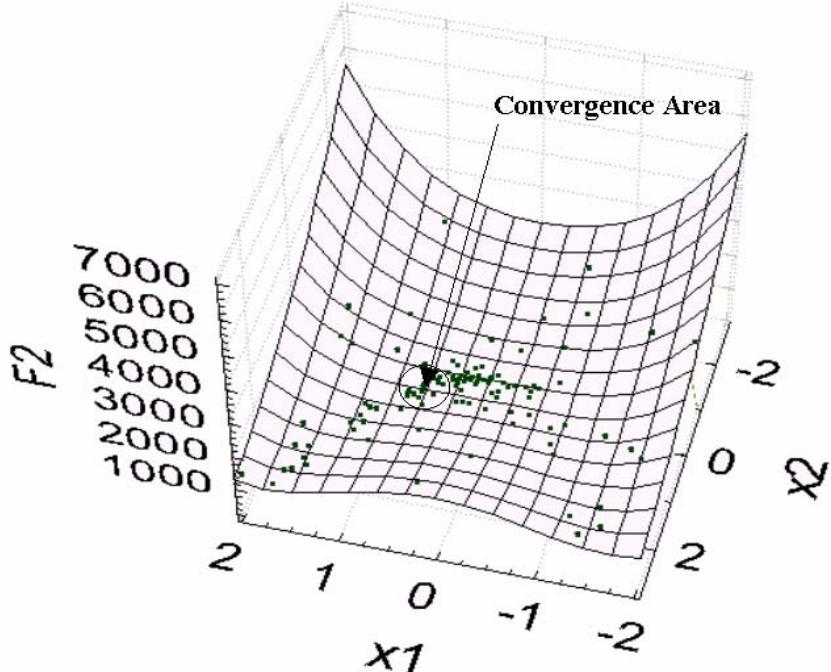


Figure 25: Dispersion of genomes during the operation of CLA-EC($L_{RP}(0.01,0.01),1,2,10$) for function F_2

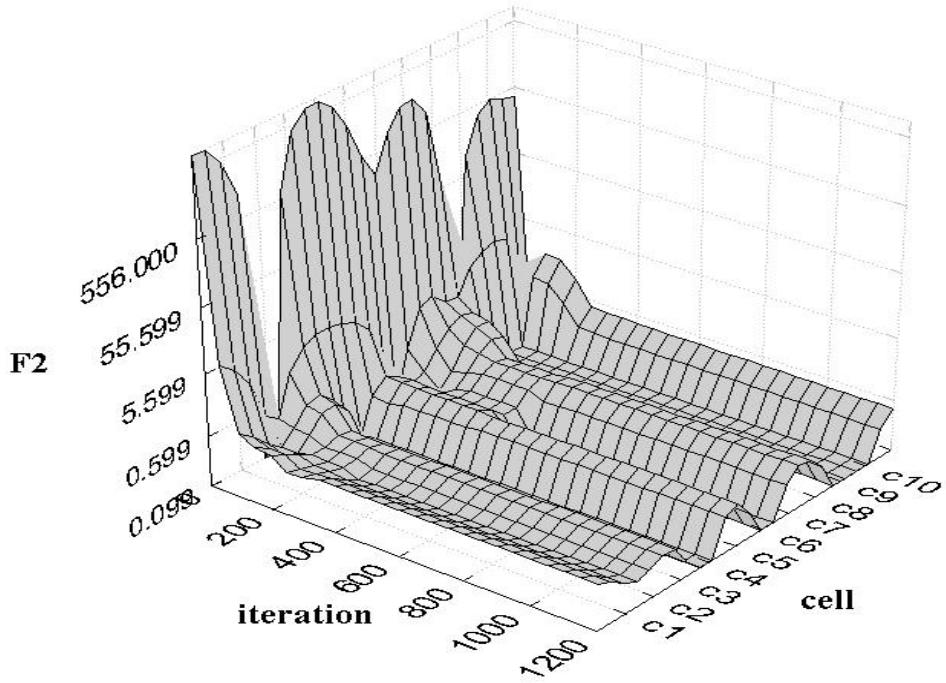
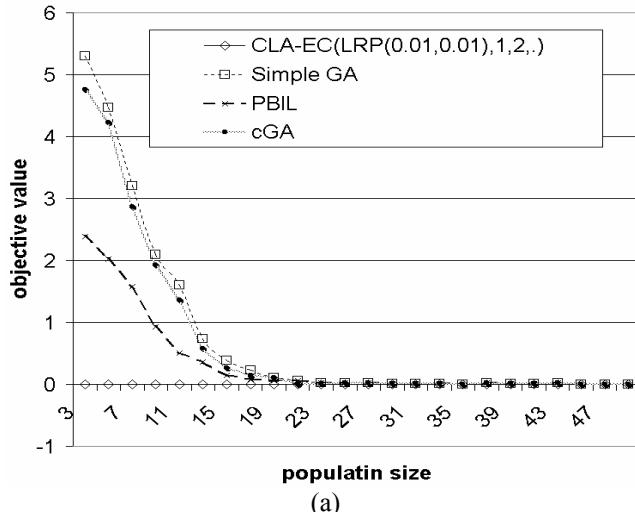


Figure 26: the evolution of the fitness of all genomes of CLA-EC($L_{RP}(0.01,0.01),1,2,10$) during a typical run for function F_2

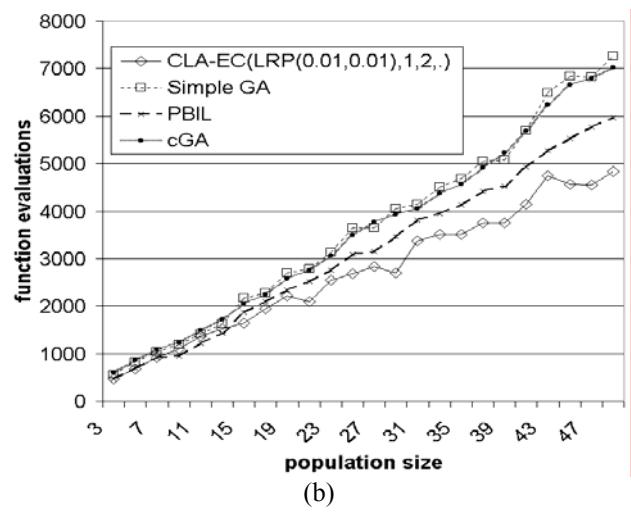
CLA-EC versus Simple Genetic Algorithm (SGA), Population based Incremental Learning (PBIL), and Compact genetic Algorithm (cGA): To show the performance of CLA-EC for real-valued function optimization, we compare it with SGA [11], cGA [13], and PBIL [2]. The SGA uses two-tournament selection without replacement and uniform crossover with exchange probability 0.5. Mutation is not used and crossover is applied with probability one. The PBIL uses learning rate 0.01 and the number of selection genomes is half of the population size. The parameters of cGA is the same as the parameters used in [13]. Convergence is considered as the termination condition for all algorithms. For CLA-EC, Se is set to 2 for functions F_1 , F_2 , F_4 , F_5 , and set to 3 for function F_3 . Results are reported in figures 27 through 31 indicating the superiority of the proposed algorithm over the SGA, PBIL and cGA.



population size

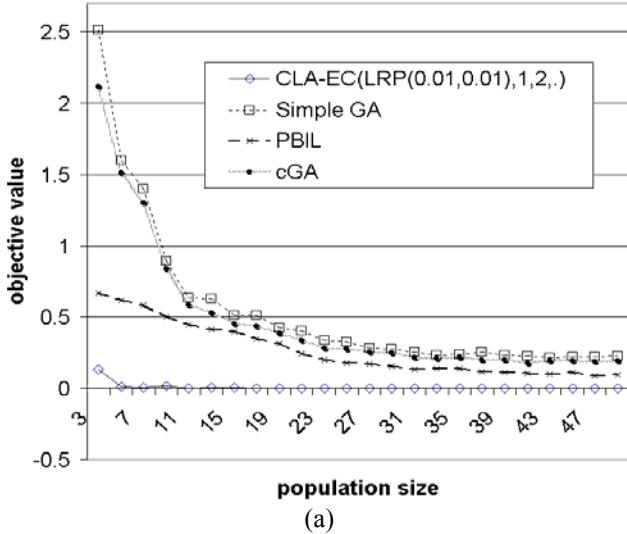
(a)

Figure 27: CLA-EC(LRP(0.01,0.01),1, 2 , -) and Simple GA for function F₁
a) Objective value



population size

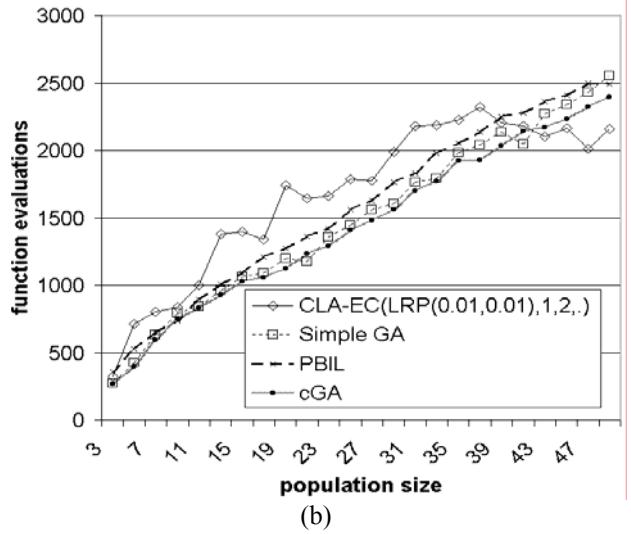
(b)



population size

(a)

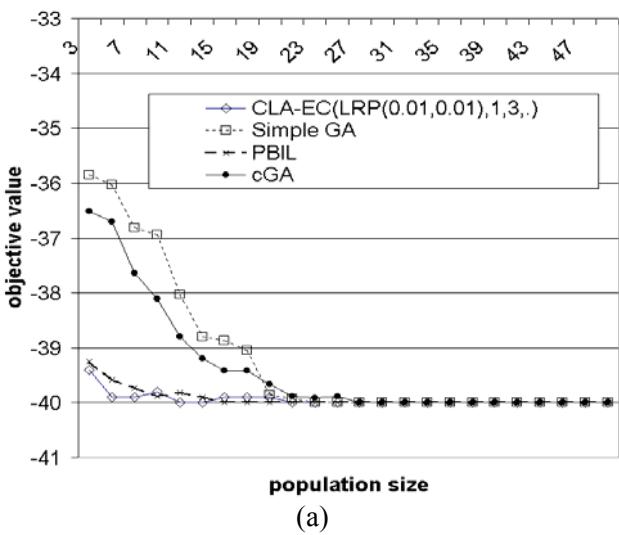
Figure 28: CLA-EC(LRP(0.01,0.01),1, 2 , -) and Simple GA for function F₂
a) Objective value



population size

(b)

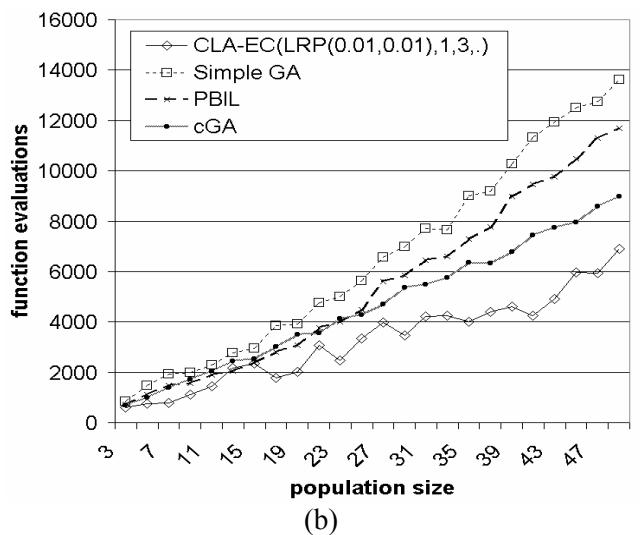
b) function evaluations



population size

(a)

Figure 29: CLA-EC(LRP(0.01,0.01),1, 3 , -) and Simple GA for function F₃
a) Objective value



population size

(b)

b) function evaluations

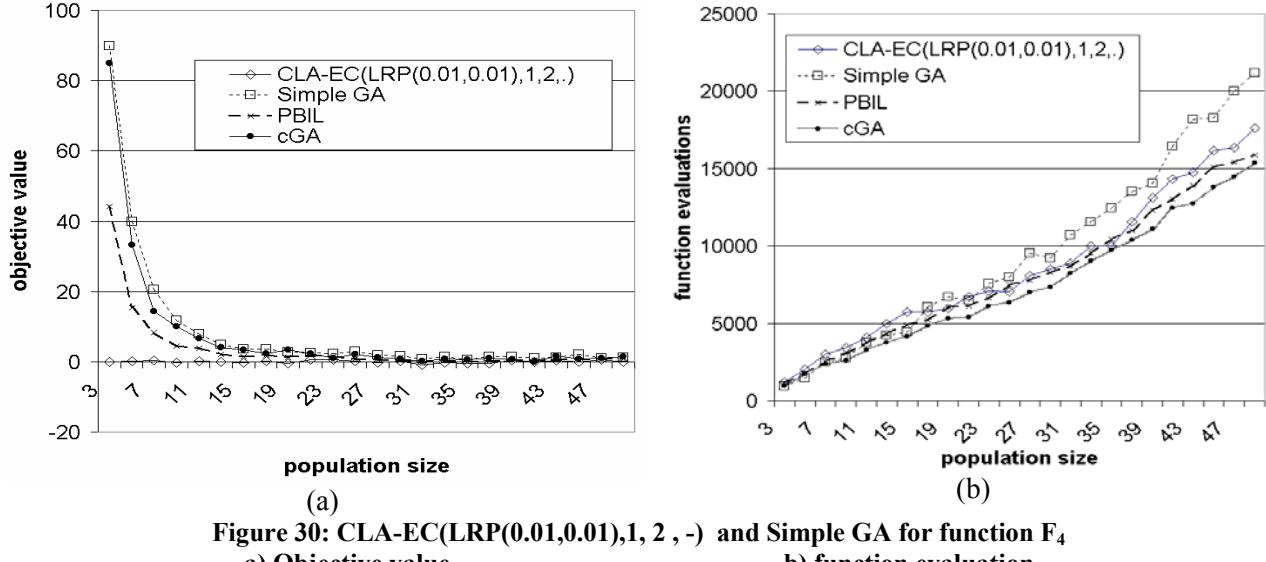


Figure 30: CLA-EC(LRP(0.01,0.01),1, 2 , -) and Simple GA for function F₄
a) Objective value **b) function evaluation**

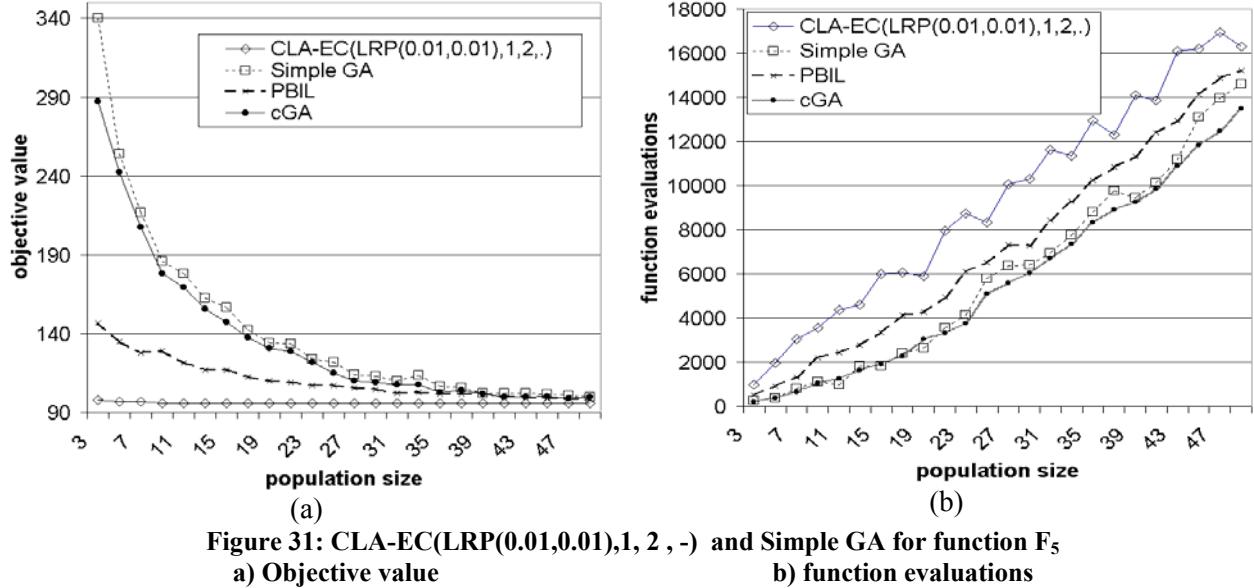


Figure 31: CLA-EC(LRP(0.01,0.01),1, 2 , -) and Simple GA for function F₅
a) Objective value **b) function evaluations**

4.2. Data clustering

In a clustering problem [9], a data set $S = \{\mathbf{x}_1, \dots, \mathbf{x}_M\}$ is given in N -dimensional Euclidean space, where $\mathbf{x}_i \in R^N$ and M is the number of data. Considering K clusters, represented by C_1, \dots, C_K , the clusters should satisfy the following conditions,

$$\begin{aligned} C_i &\neq \emptyset, i = 1, \dots, K, \\ \bigcup_{i=1}^K C_i &= S, \\ C_i \cap C_j &= \emptyset, i \neq j, i, j = 1, \dots, K. \end{aligned}$$

We propose to use CLA-EC to determine the K cluster centers of the data set in R^N ; thereby clustering the set of M points of $S = \{\mathbf{x}_1, \dots, \mathbf{x}_M\}$. The sum of the squared distances of the data from their respective clusters,

is taken as the clustering metric, and is denoted by f . The aim is to search the cluster centers in such a way that f be minimized. The proposed algorithm consists of three phases: preprocessing phase, CLA-EC phase and clustering phase.

4.2.1. Preprocessing phase

The aim of preprocessing phase is to reduce the size of the search space on which CLA-EC will operate. To reduce the size of the search space, at first the largest and the smallest values of each dimension of data set are computed as follows:

$$\begin{aligned}\min_j &= \min_{1 \leq i \leq M} \{\mathbf{x}_{i,j}\} \\ \max_j &= \max_{1 \leq i \leq M} \{\mathbf{x}_{i,j}\} \\ \Delta_j &= \max_j - \min_j\end{aligned}$$

where $\mathbf{x}_{i,j}$ is the j th component of \mathbf{x}_i . Second, $R' = [0, \Delta_1] \times \dots \times [0, \Delta_N]$ is defined.

4.2.2. The CLA-EC phase

In CLA-EC phase, clusters are optimized with respect to the squared error criterion. The characteristics of the applied CLA-EC are as follows.

String genome representation: Each string genome is represented by a binary string consisting of $M \times N$ parts where each part is a representation of an encoded real number. Let $\lambda'_{i,j}$ be $(i \times N + j)^{\text{th}}$ part of string genome, where j is the dimension of the center of cluster i in R' . If the binary representation of $\lambda'_{i,j}$ has w_{ij} bits, then in a N -dimensional space with K clusters, the length of a string genome is $m = \sum w_{ij}$.

Fitness function: To compute the fitness value of ξ , at first, we compute $\lambda'_{i,j}$ by decoding ξ , and we set $\lambda_{i,j}$ to be $(\lambda'_{i,j} + \min_j)$. The fitness value of a genome is computed as follows:

$$f(\xi) = f(\boldsymbol{\lambda}_1, \dots, \boldsymbol{\lambda}_K) = \sum_{i=1}^M (\mathbf{A}_i^*)^2,$$

where,

$$\mathbf{A}_{i,j} = \|\mathbf{x}_i - \boldsymbol{\lambda}_j\| \quad \text{And} \quad \mathbf{A}_i^* = \min_{1 \leq j \leq K} \mathbf{A}_{i,j}$$

Termination Criteria: CLA-EC stops after a pre-specified number of iterations. The best string genome found in the last iteration is the solution for the clustering problem. For the experimentations that follow the maximum number of iteration is set to 200.

4.2.3. The Clustering Phase

In this phase, the clusters are created using their centers that are encoded in the best string genome of previous phase. This is done by assigning each point \mathbf{x}_i , $i=1 \dots M$, to one of the clusters C_k with center $\boldsymbol{\lambda}_k$ such that,

$$C_k = \arg \min_{1 \leq j \leq K} \mathbf{A}_{i,j},$$

where

$$\mathbf{A}_{i,j} = \|\mathbf{x}_i - \boldsymbol{\lambda}_j\|.$$

All ties are resolved arbitrary.

Several simulations are conducted in order to evaluate the effectiveness of the proposed method. The results are then compared with the results obtained for K-means algorithm. Simulations are conducted for

nine different data sets, which we call them Data 1, Data 2, Data 3, Data 4, Data 5, Data 6, Data 7, Data 8 and IRIS Data set. The characteristics of these data sets are given below.

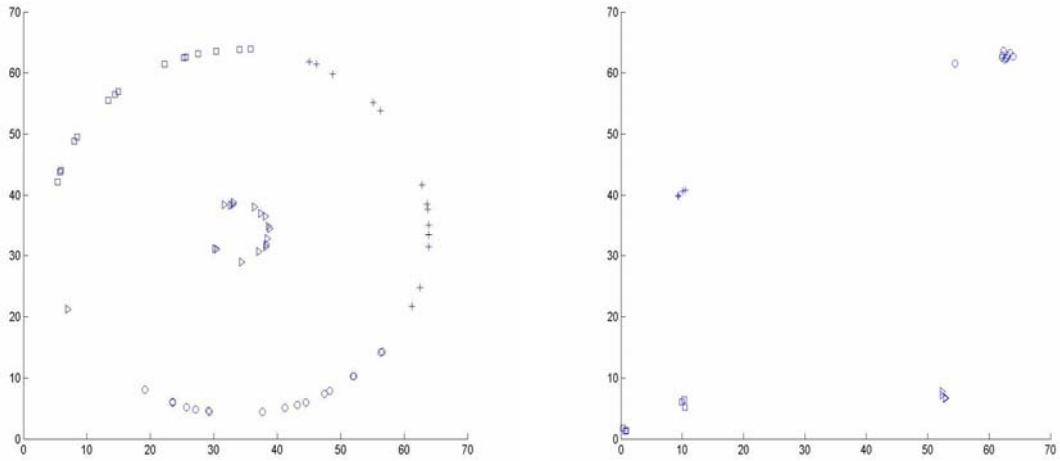


Figure 32: (a) Data 1 (b) Data 2.

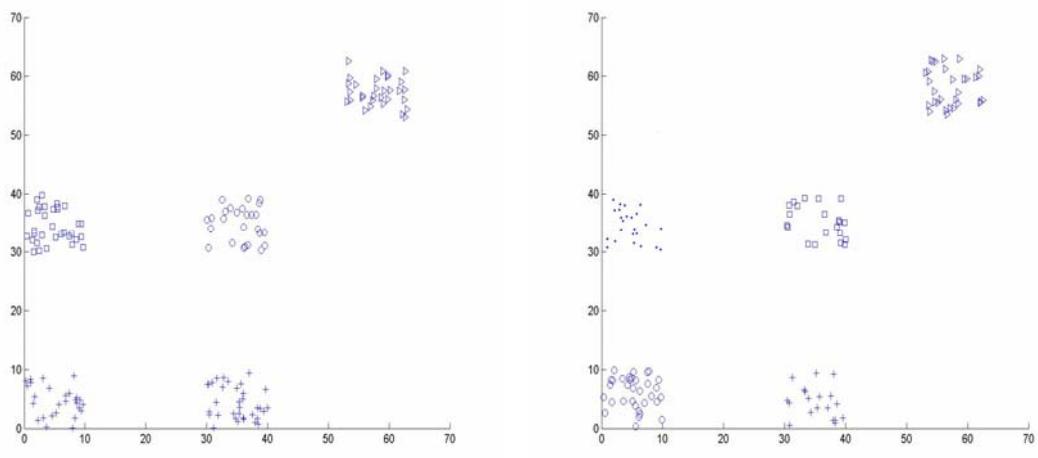


Figure 33: (a) Data 3 (b) Data 4.

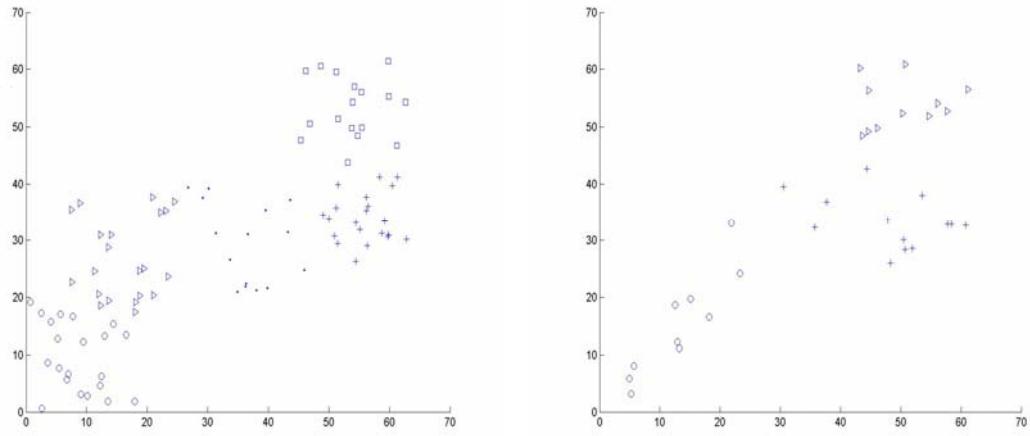


Figure 34: (a) Data 5 (b) Data 6.

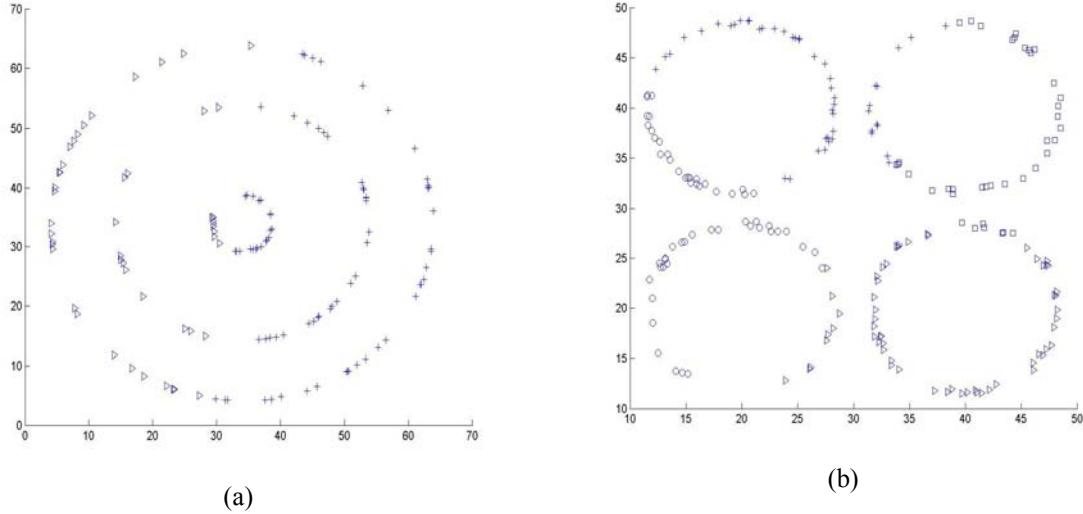


Figure 35: (a) Data 7 (b) Data 8.

Data 1: a two-dimensional data set with 4 clusters and 59 points as shown in figure 32a.

Data 2: a two-dimensional data set with 4 clusters and 25 points as shown in figure 32b.

Data 3: a two-dimensional data set with 4 clusters and 170 points as shown in figure 33a.

Data 4: a two-dimensional data set with 5 clusters and 128 points as shown in figure 33b.

Data 5: a two-dimensional data set with 5 clusters and 100 points as shown in figure 34a.

Data 6: a two-dimensional data set with 3 clusters and 35 points as shown in figure 34b.

Data 7: a two-dimensional data set with 2 clusters and 131 points as shown in figure 35a.

Data 8: a two-dimensional data set with 4 clusters and 204 points as shown in figure 35b.

Iris data: This data set represents different categories of irises having four feature values. The four feature values represent the sepal length, sepal width, petal length and the petal width in centimeter. It has 3 clusters with 150 points.

Tables 1 through 6 show the results of the CLA-EC-clustering algorithm for all data sets, Data 1 through Data 6, for different values of parameters of the CLA-EC such as the type of the learning automata, penalty and reward parameters of the learning automata, the number of selected cells and the number of cells. For each simulation the maximum number of iterations of CLA-EC is taken to be 200. Each table has 12 columns. Columns 1, 2, 3 and 4 shows the type of learning automata, reward (*a*) and penalty (*b*) parameters and the number of selected cells (*Se*), respectively. Columns 5 through 12 show the mean and the standard deviation of 50 runs for 3, 5, 10 and 15 cells, respectively. By careful inspection of the results reported in table 1 through table 6, it is found that as the number of cells increases, the mean and the standard deviation of the result decreases. Also, it has been found that, better results are obtained when each automaton uses L_{RP} or L_{REP} learning algorithm and when *Se* is set to 1.

Figures 36 and 37 show the effect of the number of cells in CLA-EC($L_{RP}(0.01,0.01),1,1,-$) on clustering Data 8 and IRIS data set. It is shown that as the number of cells increases the quality of clustering improves.

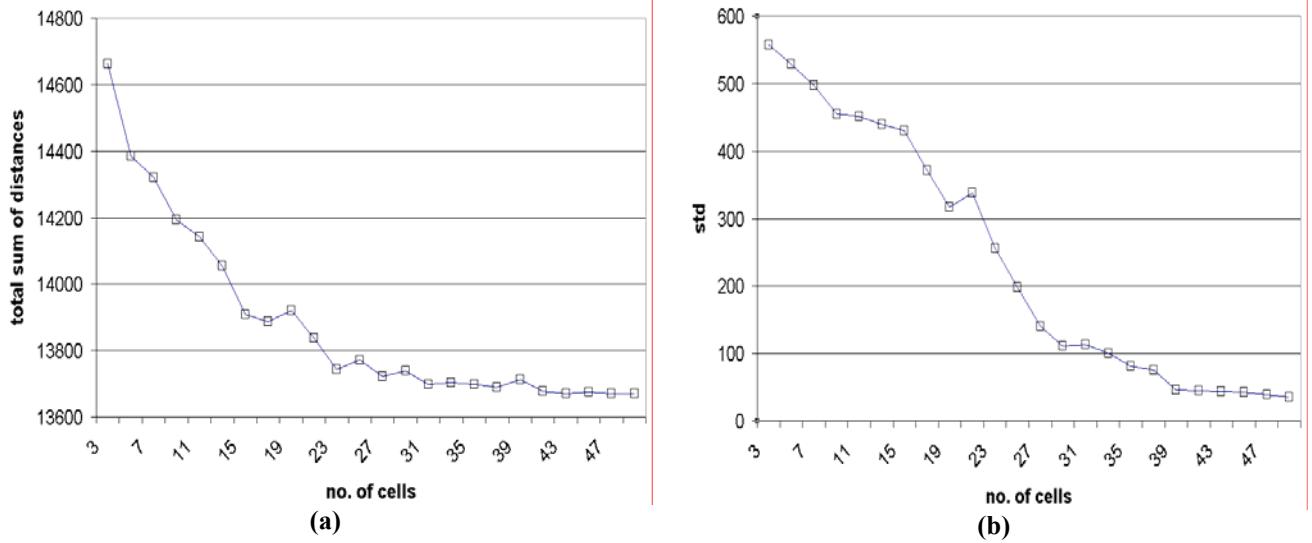


Figure 36: Effect of the number of cells on the quality of clustering for Data 8 using CLA-EC($L_{RP}(0.01,0.01),1,1,-$). (a) shows mean and (b) shows the standard deviation over 50 runs.

Figure 38 shows the fitness of the best genome (solid line) and the mean of the fitness of genomes (dashed line) for each iteration when using CLA-EC($L_{RP}(0.01,0.01),1,1,5$) for Data 1 and Data 4. Figure 39 shows the evolution of the fitness of all genomes of CLA-EC($L_{RP}(0.01,0.01),1,1,15$) during a typical run for Data 8. These figures indicate that the fitness values of all the genomes in the CLA-EC approaching a value which corresponds to a near optimal solution.

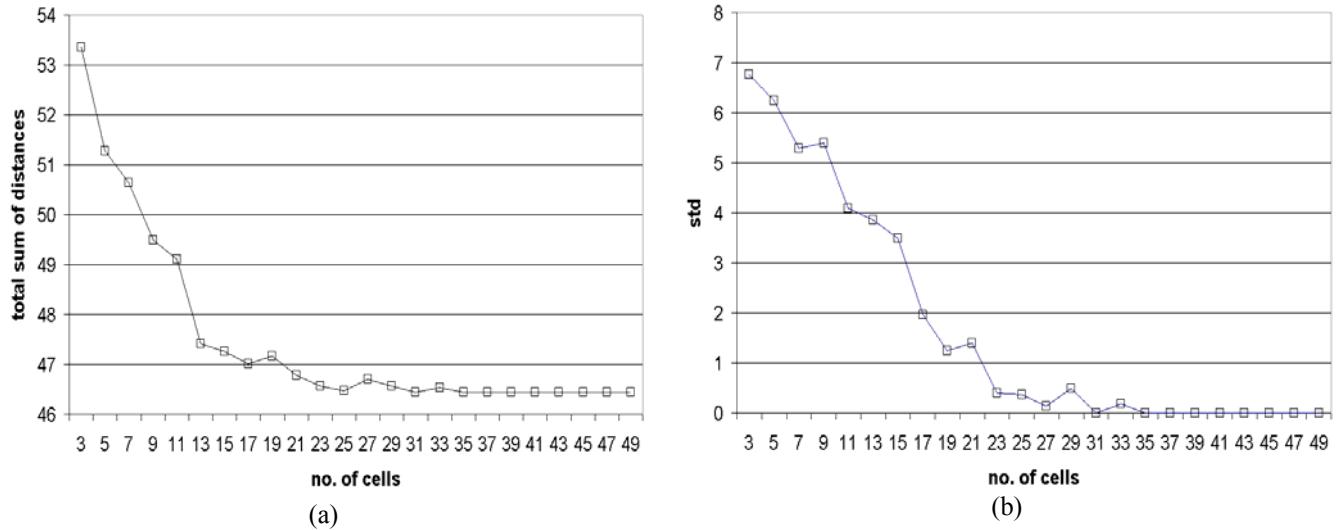


Figure 37: Effect of the number of cells on the quality of clustering for IRIS data set when using CLA-EC($L_{RP}(0.01,0.01),1,1,-$). (a) shows mean and (b) shows the standard deviation over 50 runs.

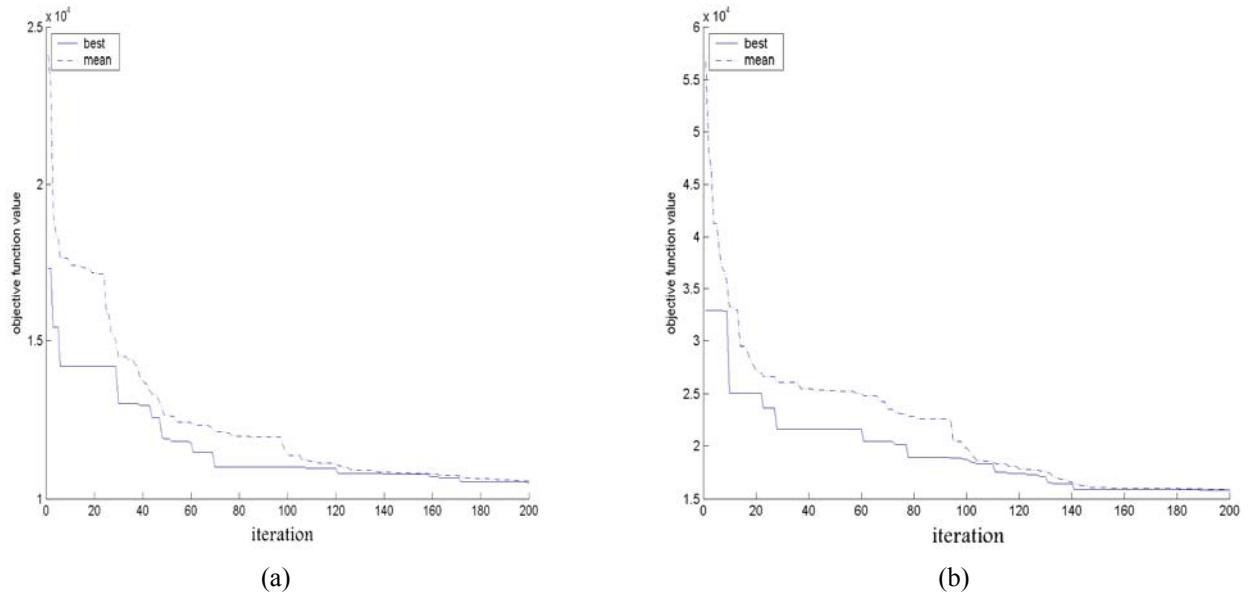


Figure 38: The fitness of the best genome (solid line) and the mean of the fitness of genomes (dashed line) for each iteration when using CLA-EC($L_{RP}(0.01, 0.01)$, 1, 1, 5) for a) Data 1 b) Data 4

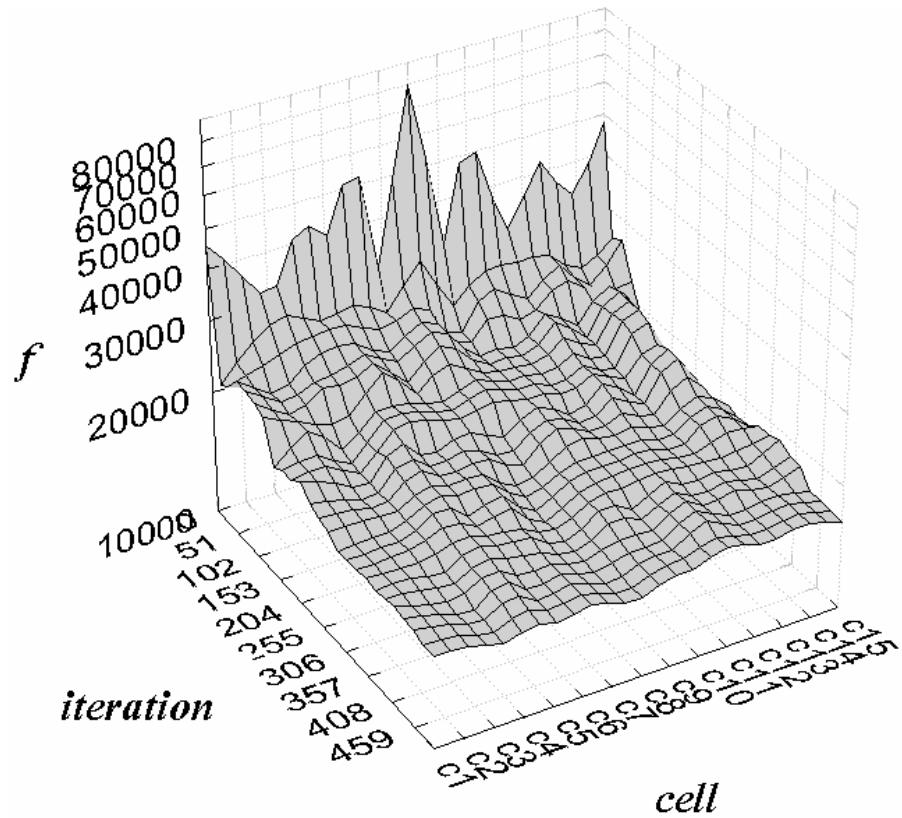


Figure 39: the evolution of the fitness of all genomes of CLA-EC($L_{RP}(0.01,0.01),1,1,15$) during a typical run for Data 8.

Table 1: The results of the CLA-EC-clustering algorithm for Data 1 (I) 3 cells (II) 5 cells (III) 10 cells (IV) 15 cells.

<i>LA</i>	<i>a</i>	<i>b</i>	<i>Se</i>	Mean (I)	Std (I)	Mean (II)	Std (II)	Mean (III)	Std (III)	Mean (IV)	Std (IV)
<i>L_{RI}</i>	0.01	0	1	10235.21	789.34	10196.34	745.45	10145.83	685.45	10113.45	663.34
	0.01	0	2	10269.56	818.25	10233.56	778.37	10195.32	738.22	10134.52	694.63
	0.01	0	3	10310.42	836.54	10297.31	810.38	10213.25	750.49	10123.25	692.56
<i>L_{RP}</i>	0.01	0.01	1	10123.23	685.12	10073.23	667.25	10067.23	614.25	10031.25	601.14
	0.01	0.01	2	10128.64	720.35	10098.66	690.26	10092.66	623.45	10043.53	612.35
	0.01	0.01	3	10319.68	789.43	10209.46	725.32	10112.35	678.41	10052.35	634.67
<i>L_{RI}</i>	0.1	0	1	10277.49	814.47	10242.38	783.45	10197.67	733.96	10163.45	656.12
	0.1	0	2	10343.23	833.48	10294.56	812.87	10284.11	753.48	10193.46	695.55
	0.1	0	3	10361.24	829.27	10317.63	823.45	10299.45	752.16	10203.65	712.24
<i>L_{ReP}</i>	0.1	0.01	1	10237.67	687.54	10113.68	610.23	10073.65	612.35	10045.62	572.35
	0.1	0.01	2	10265.59	723.95	10136.37	698.34	10093.84	634.72	10046.98	601.28
	0.1	0.01	3	10294.44	729.67	10149.43	625.41	10114.37	639.65	10072.84	593.46
<i>L_{RP}</i>	0.1	0.1	1	10243.35	675.14	10067.91	659.10	10054.45	620.23	10004.32	534.67
	0.1	0.1	2	10245.65	723.95	10132.48	711.84	10094.32	625.75	10023.47	602.31
	0.1	0.1	3	10299.92	732.33	10209.46	725.32	10134.45	643.49	10072.99	616.86

In the following experiment, we compare the results of the proposed algorithm with that of K-means [30] algorithm. For this experimentation CLA-EC has 5 cells, each automaton uses *L_{RP}* learning algorithm with $a=b=0.1$, *Se* is 1 and the maximum number of iterations is set to be 200. The results of 50 simulations for Data 1 and Data 2 are shown in figure 40, for Data 3 and Data 4 are shown in figure 41, for Data 5 and Data 6 are shown in figure 42 and for Data 7 and Data 8 are shown in figure 43. For Data 1 (figure 40a) it is found that the CLA-EC-clustering algorithm provides the optimal value of 9502.44 in 28% of the runs whereas K-means algorithm attains this value in 8% of the runs. Both algorithms get trapped at a local minimum for the other runs. For Data 2 (figure 40b) CLA-EC-clustering attains the best value of 239.10 in all the runs. K-means, on the other hand, attains this value for 28% of the runs, while in other runs it gets stuck at some of its local minima (such as 3433.77, 3497.16 and 5551.52). For Data 3 (figure 41a), Data 4 (figure 41b), Data 5 (figure 42a), Data 6 (figure 42b), Data 7 (figure 43a), Data 8 (figure 43b) and IRIS data set (figure 45) the CLA-EC-clustering attains the best values of 15545.09, 1873.71, 5525.34, 3000.43, 44100.3, 13670.63 and 46.44 in 30%, 100%, 10%, 40%, 24%, %6, %30 of the runs, respectively. The best values attained by the K-means algorithm for these data sets are 15545.09, 1873.71, 5515.34, 3000.43, 44100.3, 13670.63 and 46.44 in 20%, 30%, 2%, and 30%, %26, %14, and %24 of runs, respectively.

Table 2: The results of the CLA-EC-clustering algorithm for Data 2 (I) 3 cells (II) 5 cells (III) 10 cells (IV) 15 cells.

<i>LA</i>	<i>a</i>	<i>b</i>	<i>Se</i>	Mean (I)	Std (I)	Mean (II)	Std (II)	Mean (III)	Std (III)	Mean (IV)	Std (IV)
<i>L_{RI}</i>	0.01	0	1	340.65	53.32	239.10	0	239.10	0	239.10	0
	0.01	0	2	427.36	85.25	239.10	0	239.10	0	239.10	0
	0.01	0	3	594.96	124.39	239.10	0	239.10	0	239.10	0
<i>L_{RP}</i>	0.01	0.01	1	289.45	64.57	239.10	0	239.10	0	239.10	0
	0.01	0.01	2	394.12	161.92	239.10	0	239.10	0	239.10	0
	0.01	0.01	3	384.75	127.64	239.10	0	239.10	0	239.10	0
<i>L_{RI}</i>	0.1	0	1	378.65	115.76	239.10	0	239.10	0	239.10	0
	0.1	0	2	493.23	179.45	239.10	0	239.10	0	239.10	0
	0.1	0	3	512.46	196.29	239.10	0	239.10	0	239.10	0
<i>L_{ReP}</i>	0.1	0.01	1	239.10	0	239.10	0	239.10	0	239.10	0
	0.1	0.01	2	434.45	186.56	239.10	0	239.10	0	239.10	0
	0.1	0.01	3	486.46	208.46	239.10	0	239.10	0	239.10	0
<i>L_{RP}</i>	0.1	0.1	1	239.10	0	239.10	0	239.10	0	239.10	0
	0.1	0.1	2	387.65	113.65	239.10	0	239.10	0	239.10	0
	0.1	0.1	3	363.81	98.65	239.10	0	239.10	0	239.10	0

Table 3: The results of the CLA-EC-clustering algorithm for Data 3 (I) 3 cells (II) 5 cells (III) 10 cells (IV) 15 cells.

LA	a	b	Se	Mean (I)	Std (I)	Mean (II)	Std (II)	Mean (III)	Std (III)	Mean (IV)	Std (IV)
L_{RI}	0.01	0	1	17884.37	1635.74	17026.48	814.57	16542.37	671.57	16018.58	453.92
	0.01	0	2	18242.67	2015.13	17922.18	935.68	17922.18	773.35	16560.39	591.23
	0.01	0	3	18654.68	2135.68	18240.87	1124.95	18031.11	804.13	17024.82	582.10
L_{RP}	0.01	0.01	1	17132.17	579.52	15942.91	461.26	15799.84	201.24	15671.66	44.98
	0.01	0.01	2	17824.57	796.11	16472.36	646.79	16032.76	287.63	15755.49	73.56
	0.01	0.01	3	19367.29	1246.85	16917.84	937.64	16125.85	595.37	15843.57	123.34
L_{RI}	0.1	0	1	18521.24	1485.37	17958.54	929.48	17073.25	541.19	16103.58	440.74
	0.1	0	2	19321.39	1612.92	18227.84	1120.92	17624.32	735.85	16357.78	489.30
	0.1	0	3	20472.85	2011.46	18583.67	1242.28	18583.67	1242.28	16302.74	519.64
L_{ReP}	0.1	0.01	1	17471.17	932.55	16102.39	591.34	16953.46	524.69	15829.73	325.65
	0.1	0.01	2	18671.39	1107.67	17117.42	839.52	16793.37	637.99	16101.13	532.43
	0.1	0.01	3	18991.26	1262.58	17295.83	841.19	16851.27	541.19	15641.14	401.35
L_{RP}	0.1	0.1	1	17012.38	539.36	15889.02	413.71	15784.09	346.59	15603.09	120.08
	0.1	0.1	2	18031.13	1134.57	17230.54	962.54	16104.83	607.03	15691.14	154.45
	0.1	0.1	3	18328.75	1517.26	17530.36	1424.94	16354.57	739.70	15641.14	181.35

Table 4: The results of the CLA-EC-clustering algorithm for Data 4 (I) 3 cells (II) 5 cells (III) 10 cells (IV) 15 cells.

LA	a	b	Se	Mean (I)	Std (I)	Mean (II)	Std (II)	Mean (III)	Std (III)	Mean (IV)	Std (IV)
L_{RI}	0.01	0	1	1947.36	53.79	1873.71	0	1873.71	0	1873.71	0
	0.01	0	2	2036.43	127.93	1873.71	0	1873.71	0	1873.71	0
	0.01	0	3	2016.83	98.41	1873.71	0	1873.71	0	1873.71	0
L_{RP}	0.01	0.01	1	1913.71	25.39	1873.71	0	1873.71	0	1873.71	0
	0.01	0.01	2	1889.90	10.43	1873.71	0	1873.71	0	1873.71	0
	0.01	0.01	3	1912.43	21.43	1873.71	0	1873.71	0	1873.71	0
L_{RI}	0.1	0	1	1893.45	12.45	1873.71	0	1873.71	0	1873.71	0
	0.1	0	2	1910.76	21.48	1873.71	0	1873.71	0	1873.71	0
	0.1	0	3	1925.47	39.47	1873.71	0	1873.71	0	1873.71	0
L_{ReP}	0.1	0.01	1	1873.71	0	1873.71	0	1873.71	0	1873.71	0
	0.1	0.01	2	1873.71	0	1873.71	0	1873.71	0	1873.71	0
	0.1	0.01	3	1873.71	0	1873.71	0	1873.71	0	1873.71	0
L_{RP}	0.1	0.1	1	1873.71	0	1873.71	0	1873.71	0	1873.71	0
	0.1	0.1	2	1873.71	0	1873.71	0	1873.71	0	1873.71	0
	0.1	0.1	3	1873.71	0	1873.71	0	1873.71	0	1873.71	0

Table 5: The results of the CLA-EC-clustering algorithm for Data 5 (I) 3 cells (II) 5 cells (III) 10 cells (IV) 15 cells.

LA	a	b	Se	Mean (I)	Std (I)	Mean (II)	Std (II)	Mean (III)	Std (III)	Mean (IV)	Std (IV)
L_{RI}	0.01	0	1	5815.56	624.35	5791.56	439.24	5561.68	15.31	5541.21	12.58
	0.01	0	2	5903.45	653.28	5862.51	449.31	5632.95	86.13	5583.23	62.49
	0.01	0	3	5923.27	661.78	5899.75	487.63	5681.11	53.29	5591.26	53.29
L_{RP}	0.01	0.01	1	5739.46	426.68	5714.65	310.17	5593.27	13.03	5525.34	0
	0.01	0.01	2	5902.34	531.96	5865.87	354.22	5610.34	17.49	5525.34	0
	0.01	0.01	3	5894.62	542.84	5801.24	413.75	5652.95	32.34	5596.78	32.34
L_{RI}	0.1	0	1	5915.35	590.14	5856.81	479.75	5983.57	62.58	5599.57	62.58
	0.1	0	2	5904.26	623.57	5873.35	512.38	5701.81	101.16	5612.35	84.62
	0.1	0	3	5925.14	659.38	5891.47	504.67	5784.27	153.29	5636.26	93.29
L_{ReP}	0.1	0.01	1	5893.37	603.35	5808.52	603.35	5561.15	72.82	5534.93	10.34
	0.1	0.01	2	5889.75	652.49	5849.21	629.37	5575.21	49.83	5529.46	5.38
	0.1	0.01	3	5901.23	693.79	5876.32	633.96	5574.16	34.69	5535.16	16.38
L_{RP}	0.1	0.1	1	5791.24	345.68	5683.50	229.75	5531.27	9.80	5525.34	0
	0.1	0.1	2	5832.71	357.36	5736.15	267.28	5542.15	11.25	5525.34	0
	0.1	0.1	3	5825.43	372.97	5773.50	245.14	5548.72	18.94	5529.75	3.73

Table 6: The results of the CLA-EC-clustering algorithm for Data 6 (I) 3 cells (II) 5 cells (III) 10 cells (IV) 15 cells.

LA	a	b	Se	Mean (I)	Std (I)	Mean (II)	Std (II)	Mean (III)	Std (III)	Mean (IV)	Std (IV)
L_{RI}	0.01	0	1	3188.6	190.4	3159.2	147.29	3119.60	117.72	3066.45	80.75
	0.01	0	2	3189.1	222.4	3106.92	114.05	3048.03	53.98	3044.99	67.54
	0.01	0	3	3153.5	173.25	3117.76	110.93	3083.53	85.22	3032.52	51.31
L_{RP}	0.01	0.01	1	3080.3	97.51	3073.34	79.71	3018.47	38.63	3023.19	46.49
	0.01	0.01	2	3150.58	189.48	3076.89	90.76	3069.81	79.68	3034.13	52.03
	0.01	0.01	3	3151.82	192.67	3057.34	78.79	3036.27	61.60	3021.75	39.71
L_{RI}	0.1	0	1	3051.17	85.98	3022.63	40.33	3002.24	18.84	3001.4	14.85
	0.1	0	2	3023.73	43.55	3036.75	65.0	3012.01	27.75	3009.63	34.31
	0.1	0	3	3107.11	161.93	3034.65	69.36	3000.71	15.75	3000.52	4.69
L_{ReP}	0.1	0.01	1	3012.37	30.13	3026.18	48.01	3004.88	23.82	3001.23	18.23
	0.1	0.01	2	3056	92.41	3050.16	86.44	3019.23	41.99	3005.23	29.99
	0.1	0.01	3	3064.54	85.19	3021.80	46.73	3004.94	23.26	3000.89	13.26
L_{RP}	0.1	0.1	1	3063.85	97.97	3078.00	118.17	3022.78	42.64	3012.23	23.21
	0.1	0.1	2	3181.84	212.12	3129.22	164.77	3030.86	55.33	3023.86	46.28
	0.1	0.1	3	3554.93	514.22	3162.72	190.51	3041.28	77.07	3010.56	34.67

Table 7: The results of the CLA-EC($L_{RP}(0.1,0.1),1,1, 5$) algorithm (*maximum 200 iterations*) and the K-means algorithm for Data 1,2,3,4,5,6,7,8, IRIS - Columns ‘Mean’ and ‘Std’ show the mean and standard deviation over 50 runs.

Data Set	(CLA-EC-clustering) Mean	(CLA-EC-clustering) Std	(Kmeans) Mean	(Kmeans) Std
1	10067.92	656.10	10373.33	694.78
2	239.10	0	3980.59	3073.99
3	15889.02	413.71	19457.38	7526.98
4	1873.71	0	8473.58	5424.68
5	5683.50	229.75	5920.9	817.05
6	3078.0	118.17	3206.67	469.77
7	44514.27	485.48	44630.6	1159.91
8	14384.89	529.69	14096.57	669.02
Iris	51.27	6.25	52.96	8.37

Table 7 shows the summary of the results of this experimentation. By careful inspection of the results it is found that the CLA-EC($L_{RP}(0.1,0.1),1,1, 5$) performs better than the K-means method for Data 1, Data 2, Data 3, Data 4, Data 4, Data5, Data 6, Data 7 and IRIS data set and for Data 8 it is found that the number of cells required by CLA-EC in order to perform better than K-means must be greater than 15 (figure 44). The CLA-EC($L_{RP}(0.1,0.1),1,1,15$) attains the best values of 13670.63 in %16 of the runs.

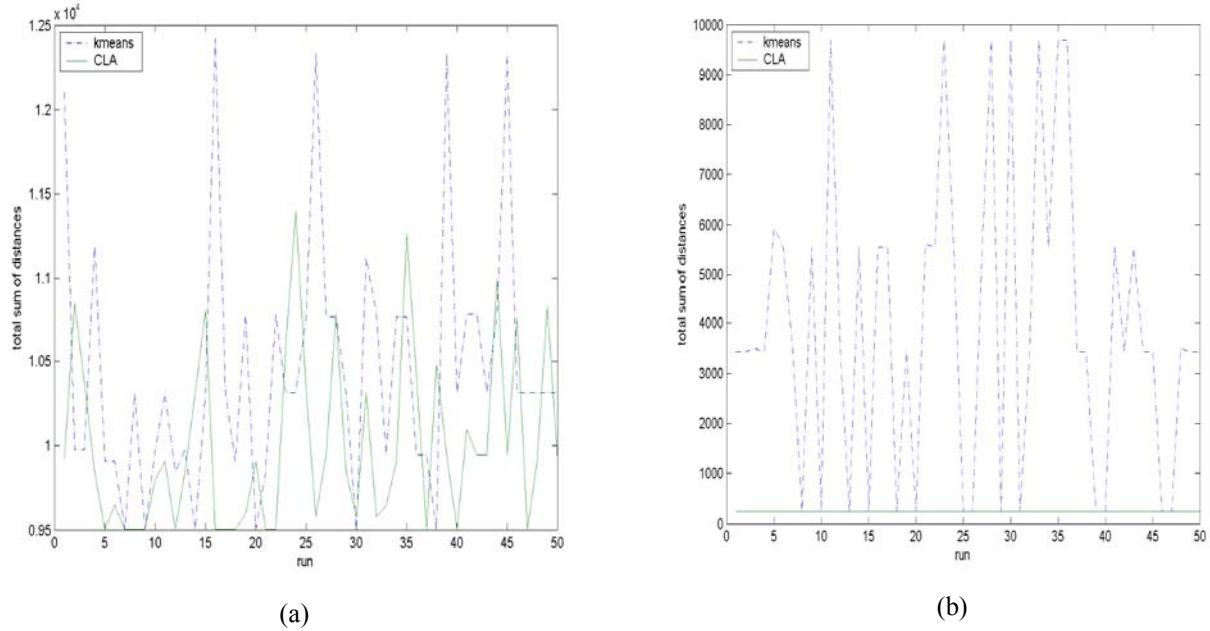


Figure 40: Comparison of the K-means and the CLA-EC($L_{RP}(0.1,0.1),1,1,5$) (a) shows the total sum of distances obtained for K-means and CLA-EC($L_{RP}(0.1,0.1),1,1,5$) over 50 different runs for Data 1- (b) shows the total sum of distances obtained for K-means and CLA-EC($L_{RP}(0.1,0.1),1,1,5$) over 50 different runs for Data 2. The solid line is for the CLA-EC($L_{RP}(0.1,0.1),1,1,5$) and the dashed line is for the K-means algorithm.

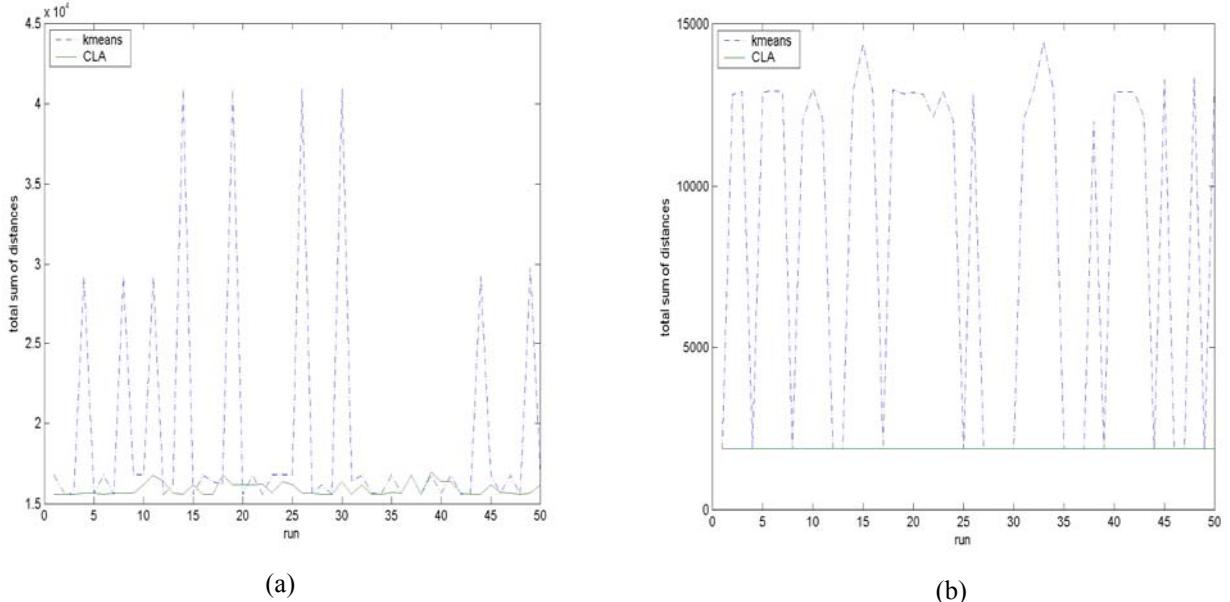


Figure 41: Comparison of the K-means and the CLA-EC($L_{RP}(0.1,0.1),1,1,5$) (a) shows the total sum of distances obtained for K-means and CLA-EC($L_{RP}(0.1,0.1),1,1,5$) over 50 different runs for Data 3- (b) shows the total sum of distances obtained for K-means and CLA-EC($L_{RP}(0.1,0.1),1,1,5$) over 50 different runs for Data 4. The solid line is for the CLA-EC($L_{RP}(0.1,0.1),1,1,5$) and the dashed line is for the K-means algorithm.

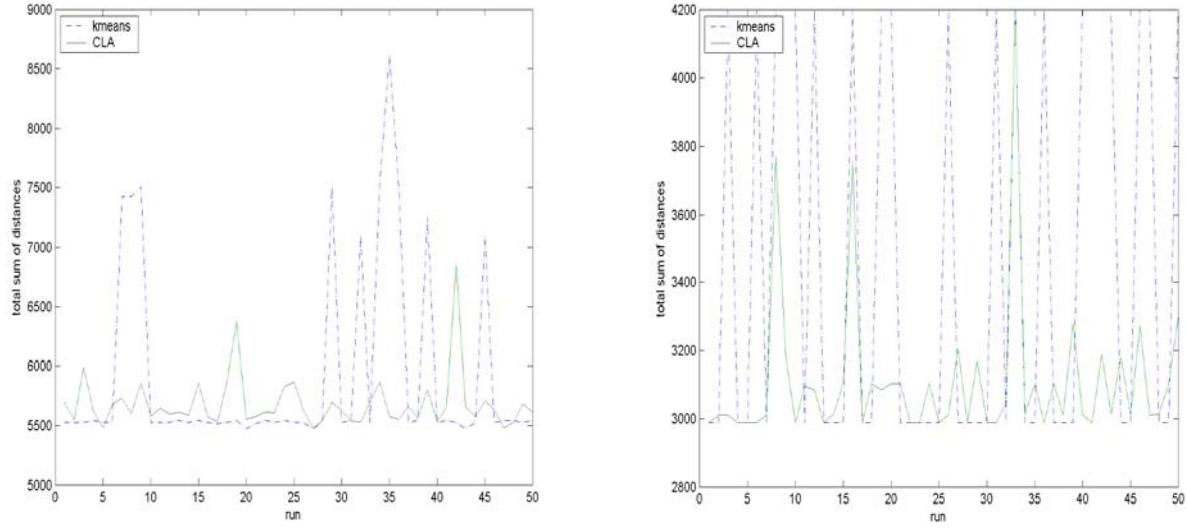


Figure 42: Comparison of the K-means and the CLA-EC($L_{RP}(0.1,0.1),1,1, 5$) (a) shows the total sum of distances obtained for K-means and CLA-EC($L_{RP}(0.1,0.1),1,1, 5$) over 50 different runs for Data 5- (b) shows the total sum of distances obtained for K-means and CLA-EC-clustering over 50 different runs for Data 6. The solid line is for the CLA-EC-clustering and the dashed line is for the K-means algorithm.

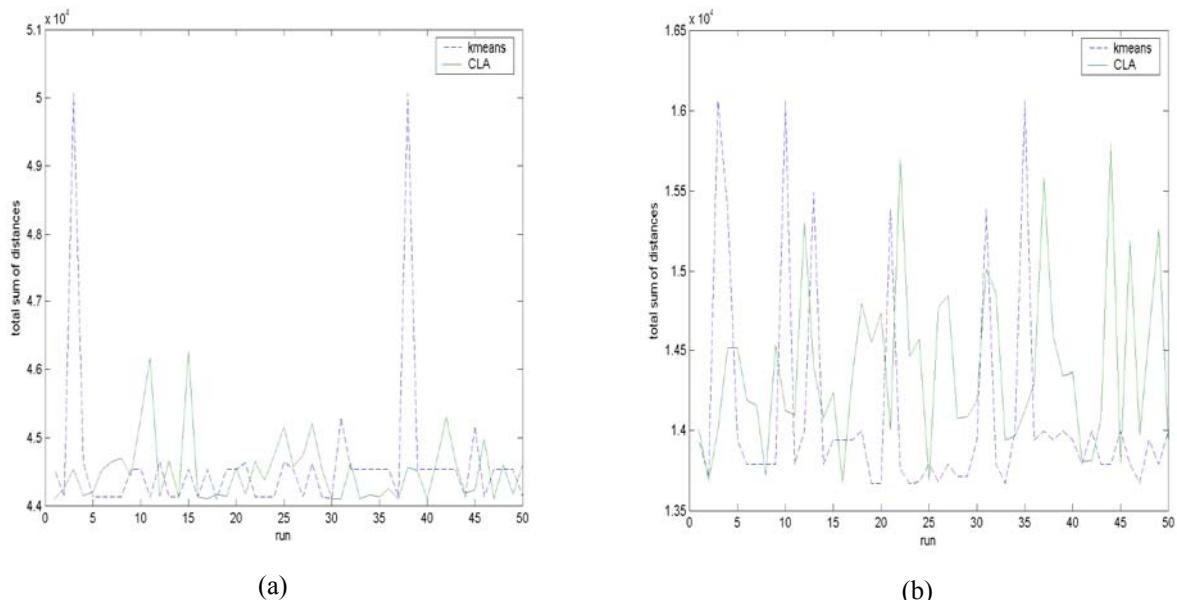


Figure 43: Comparison of the K-means and the CLA-EC($L_{RP}(0.1,0.1),1,1, 5$) (a) shows the total sum of distances obtained for K-means and CLA-EC($L_{RP}(0.1,0.1),1,1, 5$) over 50 different runs for Data 7 (b) shows the total sum of distances obtained for K-means and CLA-EC-clustering over 50 different runs for Data 8. The solid line is for the CLA-EC($L_{RP}(0.1,0.1),1,1, 5$) and the dashed line is for the K-means algorithm.

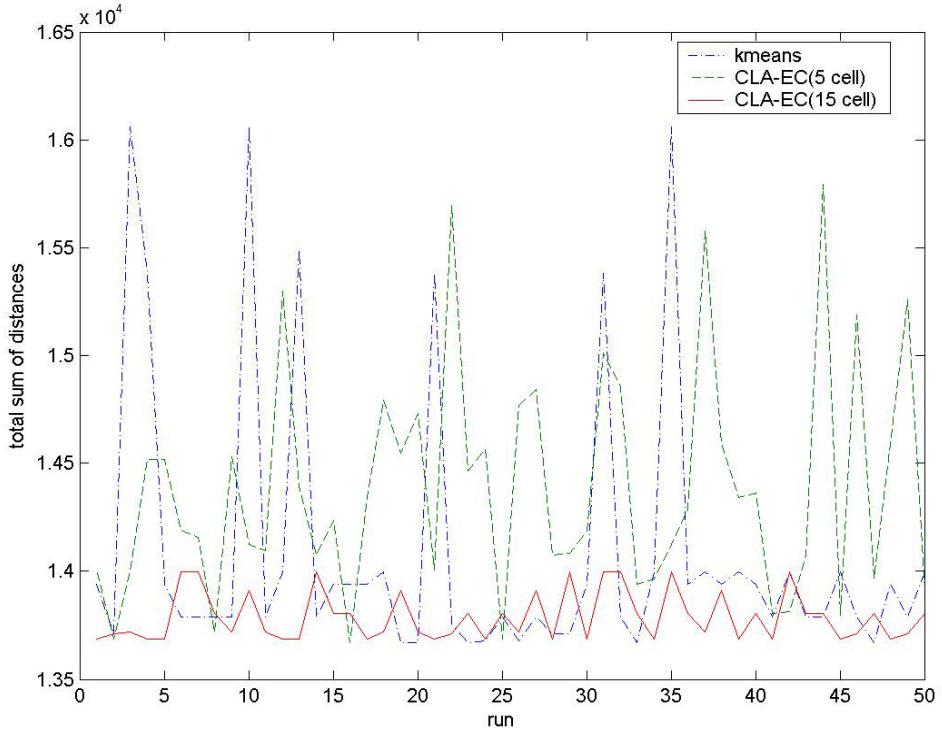


Figure 44: Comparison of the K-means and the CLA-EC($L_{RP}(0.1,0.1), 1, 1, 5$) (a) shows the total sum of distances obtained for K-means and CLA-EC($L_{RP}(0.1,0.1), 1, 1, 5$) in 50 different runs for Data 8.

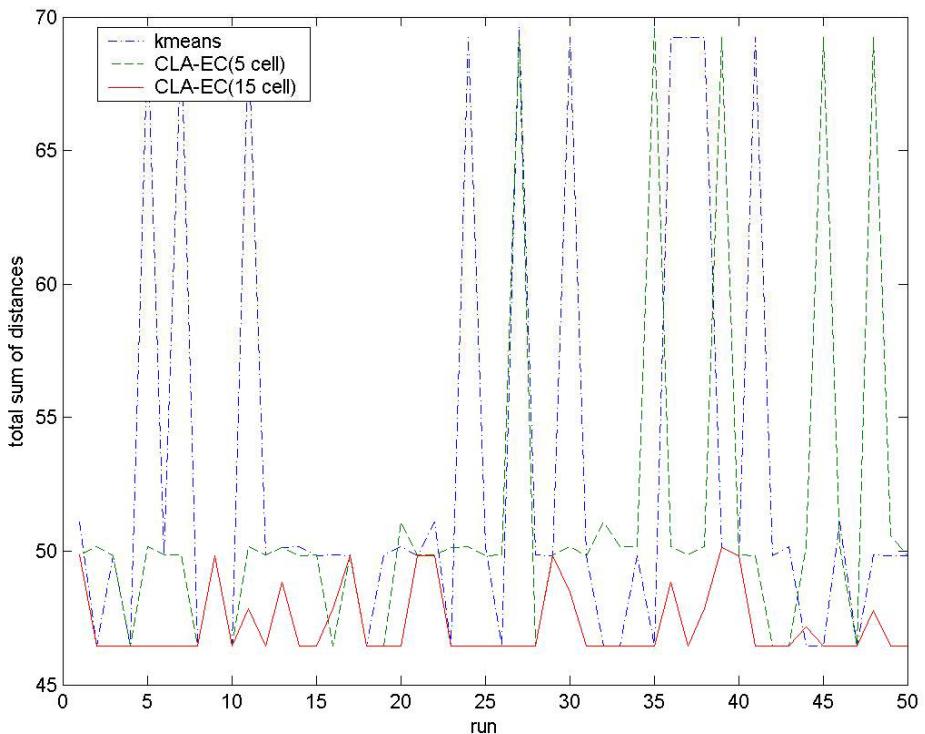


Figure 45: Comparison of the K-means and the CLA-EC($L_{RP}(0.1,0.1), 1, 1, 5$) (a) shows the total sum of distances obtained for K-means and CLA-EC($L_{RP}(0.1,0.1), 1, 1, 5$) in 50 different runs for IRIS data set.

5. Conclusion

In this paper, a new fine-grained PEAs, called CLA-EC, was proposed. This model is a combination of a model called cellular learning automata (CLA) and the evolutionary model. Like other fine-grained EAs, in CLA-EC, each cell contains a string genome, which is sampled from the search space. The cells update their states synchronously on discrete steps according to a local rule (like cellular automata) and a learning algorithm. The next state of each cell depends on the current states of its neighboring cells and the cell itself., To show the effectiveness of the proposed method it has been used to solve two optimization problems: real valued function optimization and clustering problems. The result of experiments showed that CLA-EC is a good candidate for solving optimization problems.

References

- [1] Alba, E., and Troya, J. M., "Analyzing Synchronous and Asynchronous Parallel Distributed Genetic Algorithms", Future Generation Computer Systems, Vol. 17, PP. 451-465, 2001.
- [2] Baluja, S., Caruana, R., "Removing The Genetics from The Standard Genetic Algorithm", In Proceedings of ICML'95, PP. 38–46, Morgan Kaufmann Publishers, Palo Alto, CA, 1995.
- [3] Baluja S., "A massively distributed parallel genetic algorithm". Technical Report No. CMU-CS-92-196R, Carnegie Mellon University, Pittsburgh, PA, 1992.
- [4] Baluja S., "Structure and Performance of Fine-Grain Parallelism in Genetic Search", Proceedings of the Fifth International Conference on Genetic Algorithms, pp. 155–162, Morgan Kaufmann (San Mateo, CA), 1993.
- [5] Baradaranhshemi, A, Beigy, H., and Meybodi, M. R., "Dynamic Call Access Control for Cellular Mobile Networks", Proceedings of 9th Annual CSI Computer Conference, Computer Engineering Department, Sharif University, Tehran, Iran, pp. 440-446, Feb. 2004.
- [6] Beigy, H., and Meybodi, M. R., "A Mathematical Framework for Cellular Learning Automata", Advances in Complex Systems, Vol. 7, No. 3 & 4, pp. 295-319, September 2004.
- [7] Beigy, H., and Meybodi, M. R., "A Self-Organizing Channel Assignment Algorithm: A Cellular Learning Automata Approach", Vol. 2690 of Springer-Verlag Lecture Notes in Computer Science, PP. 119-126, Springer-Verlag, 2003.
- [8] 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.
- [9] Fukunaga, K., Introduction to Statistical Pattern Recognition, Academic Press, New York, 1990.
- [10] Ghanbari, R. and Meybodi, M. R., "Load Balancing in Grid Computing Using Cellular Learning automata", Technical Report, Computer Engineering Department, Amirkabir University, Tehran, Iran, 2004.
- [11] Goldberg, D. E., Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, New York, 1989.
- [12] Gorges S. M., "Explicit Parallelism of genetic Algorithms through Population Structures", Parallel Problem Solving from Nature, p. 150–159, Springer-Verlag (Berlin), 1991.
- [13] Harik, G. R., Lobo, F. G., and Goldberg, D. E., "The Compact Genetic Algorithm", IEEE Transaction on Evolutionary Computing, Vol. 3, No. 4, PP. 287–297, 1999.
- [14] Howell, M. N., Gordon, T. J., and Brandao, F. V., "Genetic Learning Automata for Function Optimization", IEEE Transaction on Systems, Man, and Cybernetics-Part B: Cybernetics, Vol. 32, No. 6, 2002.
- [15] Khojasteh, M. R. and Meybodi, M. R. "Cooperation in Multi-Agent Systems Using Learning Automata", Iranian Journal of Electrical and Computer Engineering, Vol. 1, No. 2, pp.81-91, 2004.
- [16] Manderik B., Spiessens P., "Fine-grained parallel genetic algorithms", Proceedings of the Third International Conference on Genetic Algorithms, pp. 428–433, Morgan Kaufmann (San Mateo, CA), 1989.
- [17] Meybodi, M. R., and Kharazmi, M. R., "Application of Cellular Learning Automata to Image Processing", Journal of Amirkabir, Vol. 14, No. 56A, pp. 1101-1126, 2004.
- [18] Meybodi, M. R., and Khojaste, M. R., "Application of Cellular Learning Automata in Modeling of Commerce Networks", in Proceedings of 6th Annual International Computer Society of Iran Computer Conference CSICC2001, Isfahan, Iran, PP. 284-295, 2001.

- [19] Meybodi, M. R., and Mehdipour, F., "VLSI Placement Using Cellular Learning Automata", in Proceedings of 8th Annual International Computer Society of Iran Computer Conference CSICC2001, Mashhad, Iran, PP. 195-203, 2003.
- [20] Meybodi, M. R., and Taherkhani, M., "Application of Cellular Learning Automata to Modeling of Rumor Diffusion", in Proceedings of 9th Conference on Electrical Engineering, Power and Water institute of Technology, Tehran, Iran, PP. 102-110, May 2001.
- [21] Meybodi, M. R., and Kharazmi, M. R., "Image Restoration Using Cellular Learning Automata", in Proceedings of the Second Iranian Conference on Machine Vision, Image Processing and Applications, Tehran, Iran, PP. 261-270, 2003.
- [22] Meybodi, M. R., Beigy, H., and Taherkhani, M., "Cellular Learning Automata", in Proceedings of 6th Annual International Computer Society of Iran Computer Conference CSICC2001, Isfahan, Iran, PP. 153-163, 2001.
- [23] Mühlenbein, H., "Evolution in time and space-The parallel genetic algorithm", Foundations of Genetic Algorithms, pp. 316–337, Morgan Kaufmann (San Mateo, CA), 1991.
- [24] Mühlenbein, H., "Parallel genetic algorithms, population genetics and combinatorial optimization", Proceedings of the Third International Conference on Genetic Algorithms, pp. 416–421, Morgan Kaufmann (San Mateo, CA), 1989.
- [25] Munetomi, M., Takai, Y., and Sato, Y., "StGA: An Application of Genetic Algorithm to Stochastic Learning Automata", Syst. Comput. Jpn., Vol. 27, PP. 68-78, 1996.
- [26] Narendra, K. S., and Thathachar, M. A. L., *Learning Automata: An Introduction*, Printice-Hall Inc, 1989.
- [27] Rastegar, R., and Meybodi, M. R., "Evolutionary Algorithm: A Cellular Learning Automata Approach", Technical Report, Soft-Computing Lab, Computer Engineering Department, Amirkabir University, Tehran, Iran, 2003.
- [28] Rastegar, R., and Meybodi, M. R., "LAEDA: A New Evolutionary Algorithm using Learning Automata", in proceeding of 9th Annual International Computer Society of Iran Computer Conference CSICC2004, Tehran, Iran, pp. 456-464, 2004.
- [29] Rudolph, G., and Joachim, S., "A Cellular Genetic Algorithm with Self-Adjusting Acceptance Threshold", Genetic Algorithms in Engineering Systems: Innovations and Applications, Conference Publication, No. 414, PP. 365-372, 1995.
- [30] Selim, S. Z., and Ismail, M. A., "K-means-type Algorithm: Generalized Convergence Theorem and Characterization of Local Optimality", IEEE Transaction on Pattern Analysis and Machine Intelligence, Vol 6, PP 81-87, 1984.
- [31] Thathachar, M. A. L., and Sastry, P. S., "Varieties of Learning Automata: An Overview", IEEE Transaction on Systems, Man, and Cybernetics-Part B: Cybernetics, Vol. 32, No. 6, PP. 711-722, 2002.
- [32] Tomassini, M., "The Parallel Genetic Cellular Automata: Application to Global function Optimization", in proceedings of International Artificial Neural Nets and Genetic Algorithms Conference, Austria, PP. 385-391, Springer, Wein, 1993.
- [33] Whitley, D. "Cellular genetic algorithms", Proceedings of the Fifth International Conference on Genetic Algorithms, San Mateo, CA: Morgan Kaufmann Publishers, pp. 658, 1993.
- [34] Wolfram, S., *Cellular Automata and Complexity*, Perseus Books Group, 1994.