International Journal of Hybrid Intelligent Systems 3 (2006) 83–98
IOS Press

A new fine-grained evolutionary algorithm based on cellular learning automata

Reza Rastegar[a,*], Mohammad Reza Meybodi[b] and Arash Hariri[c]
[a]*Department of Mathematics, Southern Illinois University, Carbondale, IL 62901, US*
[b]*Department of Computer Engineering, Amirkabir University of Technology, Tehran, Iran*
[c]*Department of Electrical and Computer Engineering, Shahid Beheshti University, Tehran, Iran*

Abstract. In this paper a new evolutionary algorithm, called the CLA-EC (Cellular Learning Automata Based Evolutionary Computing), is proposed. This algorithm is a combination of evolutionary algorithms and the Cellular Learning Automata (CLA). In the CLA-EC each genome string in the population is assigned to one cell of the CLA, which is equipped with a set of learning automata. Actions selected by the learning automata of a cell determine the genome 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. Each learning automaton in the cell updates its internal structure according to a learning algorithm and the received signal vector. The processes of action selection and updating the internal structures of learning automata are repeated until a predetermined criterion is met. To show the efficiency of the proposed model, to solve several optimization problems including real valued function optimization and data clustering problems.

Keywords: Cellular learning automata, evolutionary algorithm, learning automata, cellular automata, optimization, CLA-EC

1. Introduction

Evolutionary algorithms (EAs) form a class of random search algorithms in which the principles of natural evolution are regarded as rules for optimization. They attempt to find optimal solutions to a problem by manipulating a population of candidate solutions. The population is evaluated and the best solutions are selected for mating to form the next generation. Over a number of generations, good traits dominate the population, which results in an increase in the quality of solutions. Evolutionary algorithms are often applied to optimization problems, where well-known techniques such as gradient based algorithms, linear programming, or dynamic programming are not available, or standard methods fail to give reasonable answers due to multimodality, non-differentiability, or discontinuity of the problem at hand [1]. The basic mechanism in the 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 get mixed by recombination (crossover) to form better individuals. Mutation also exists in the EAs, but it is considered as a secondary operator. Its role is to ensure that diversity is not lost in the population; so, the EAs can continue to explore.

EAs can be parallel algorithms. The basic idea behind the most of the parallel EAs (PEA) is dividing their tasks into chunks, and then solving the chunks simultaneously by using multiple processors. This divide-and-conquer approach can be used in the EAs in many different ways, and literature contains many cases of successful parallel implementations. Some parallelization methods use a single population, while the others divide the population into several relatively isolated subpopulations. Some methods can exploit massively parallel computer architectures, while the others are better suited for multi-computers with fewer, but more powerful processing elements. There are four types of PEAs: the global single-population master-slave EAs,

*Corresponding author. Tel.: +1 618 529 3750; E-mail: rrastegar@ieee.org.

the single-population fine-grained PEA, the multiple-population coarse-grained EAs, and the hierarchical combinations [18]. In a master-slave PEA, there is a single panmictic population, but the fitness evaluation is distributed among several processors. Since in this type of EAs selection and crossover consider the entire population, it is also known as the global PEAs. Fine-grained 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 and this allows some interactions among all of the individuals. The ideal case is to have only one individual for each available processor. The multiple-population PEAs are more sophisticated, as they consist of several subpopulations that exchange individuals occasionally. This exchange of individuals is called migration, and it is controlled by several parameters. A fine-grained PEA [2,14,16] has only one population, but it has a spatial structure which limits the interactions between individuals. In this type, an individual can only compete and mate with its neighbors, but since neighborhoods overlap, good solutions may be disseminated across the entire population. It is common to place the individuals of a fine-grained PEA on a 1, 2, or 3-dimensional regular grid. Whitley [2] has shown that this scheme can be modeled by using the Cellular Automata (CA) with stochastic rules. The CA model is an abstract model, which consists of a large number of simple identical cells with local interactions. The CA model is a non-linear dynamical system, in which space and time are discrete. It is cellular, because it is made up of some cells like the points in a lattice or like the squares of a checker board, and it is called automata, because it follows a simple rule [19]. It is specifically suitable for modeling the natural systems, which can be described as massive collections of simple objects locally interacting with each other. Informally, a d-dimensional CA model consists of a finite or infinite $d-$dimensional lattice of identical cells. Each cell can assume a state (a 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, including local selection, crossover, and mutation. The new state of each cell depends on the previous states of a set of cells including the cell itself and its neighbors. Decentralization is a feature of the CA (fine-grained PEAs) in which due to the large spatial separation of cells (genomes of fine-grained PEAs) or the limited bandwidth of communication channels, complete information exchange may not be feasible. Each cell in the CA can gather limited information about the other cells and the overall system. Since individual cells with partial information should make decision about state changing, an uncertainty is introduced into decisions. Learning in these decisions can overcome the introduced uncertainty [9].

Learning Automata are general-purpose stochastic optimization tools, which have been developed as models of learning systems [9,10]. They are typically used as the basis of learning systems, and through interactions with an unknown stochastic environment, they learn the optimal action for that environment. Iteratively, a learning automaton tries to choose the optimal action from a finite number of actions to apply to the environment. The environment returns a reinforcement signal, which 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.

The Cellular Learning Automata model (CLA), which has been recently introduced, is a combination of the CA and the LA [4,6]. This model is superior to the CA because of its ability to learn, and also it is superior to a single learning automaton, because it is a collection of LA, which can interact with each other to solve a particular problem. The basic idea of the CLA, which is a super class of the stochastic CAs, is to use the LA to adjust the state transition probability of the stochastic CA [4]. So far, the CLA has been used in many applications such as channel assignment in cellular mobile systems [5] and load balancing in computational grids [17].

Here, a new fine-grained PEA called the Cellular Learning Automata based Evolutionary Computing (CLA-EC) is proposed to overcome the uncertainty existing in the other fine-grained PEAs. The CLA-EC is obtained by combining the CLA model with the evolutionary algorithms. In the CLA-EC, each genome in the population is assigned to a cell of the CLA, and each cell in the CLA is equipped with a set of LA. The actions selected by the set of LA determine 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 that cell. According to the received signal, each learning automaton updates its internal structure by using a learning algorithm. The process of action selection and updating the internal structure of the learning automata is repeated until a predetermined criterion is met. The rest of this paper is organized as follows. Section 2 briefly describes the LA and the CLA. Section 3 introduces the CLA-EC. The experimental results are given in Section 4. Finally Section 5 concludes.

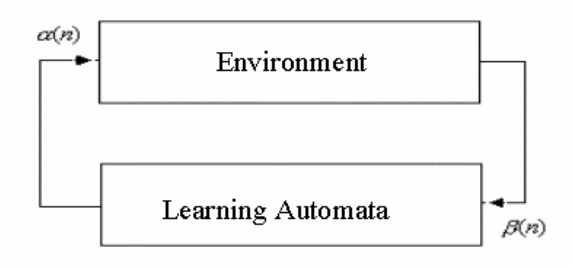


Fig. 1. The interaction between the learning automata and the environment.

2. Cellular Learning Automata

Learning Automata (LA) are adaptive decision-making devices, which operate on unknown stochastic environments. A Learning Automaton has a finite set of actions, and each action has a certain probability (unknown for the automaton) to be 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 interactions with the environment. If the learning algorithm is chosen properly, then the iterative process of interaction with the environment can result in selection of an optimal action.

Figure 1 illustrates how a stochastic automaton works in a feedback connection with a stochastic environment. Learning Automata can be classified into two main families: fixed structure and variable structure [9,10]. A variable structure learning automaton is a 4-tuple $< \alpha, \beta, p, T(\alpha, \beta, p) >$, where α and β are an action set of s actions and an environment response set respectively. The probability set p contains s probabilities, where each of them is the probability of performing the associated action in the current internal automaton state. Finally, T is the reinforcement algorithm, which modifies the action probability vector p respecting the performed action and the received response. Let a variable structure learning automaton operate in an environment with $\beta = \{0,1\}$, and let $n \in N$, where N is the set of nonnegative integers. A general linear schema for updating the action probabilities can be represented as follows. Let the action i be performed. If $\beta(n) = 0$ (reward),

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

and if $\beta(n) = 1$ (penalty),

$$\begin{aligned} p_i(n+1) &= (1-b)p_i(n) \\ p_{j \neq i}(n+1) &= (b/s-1) + (1-b)p_j(n) \end{aligned} \tag{2}$$

where a and b are reward and penalty parameters respectively. When $a = b$, the learning algorithm is called L_{RP}. If $b = 0$, it is called L_{RI} and if $0 < b << a <1$, it is called $L_{R\varepsilon P}$. Figure 2 shows the working mechanism of the variable structure learning automaton.

The potential of LA can be completely realized when multiple automata interact with each other. Interaction may assume different forms such as a tree, mesh, or array, and depending on the problem, one of these structures may be used. In most applications, full interaction among all LA is neither necessary nor natural. On the other hand, cellular automata model [19] is a mathematical model for the systems, which consist of a large number of simple identical components with local interactions. Cellular Learning Automata model (CLA), like CA, consists of a large number of simple components, which have learning capability, and act together to solve a particular problem. The CLA model is a cellular automata model in which each cell is equipped with a set of LA. The LA use their action probabilities to determine the state of that cell. Like the CA, there is a rule, under which the CLA operates. This rule and the actions selected by the neighboring learning automata determine the reinforcement signal for the learning automata residing in a cell.

The operation of cellular learning automata can be described as follows: At each step, each learning automaton, residing in a cell, chooses an action based on its action probability vector. Initially, an automaton chooses its action either on the basis of its past experience or at random. Then the rule of the cellular automata determines the reinforcement signal for the learning automaton residing in that cell. Finally, the learning automaton updates its action probability vector on the basis of the supplied reinforcement signal, chosen action, and the learning algorithm. The process of action selection and probability vector updating is continued until the desired result is obtained. It is useful to note that a CLA model is synchronous if all cells are synchronized with a global clock and activated at the same time, and it is uniform if every cell uses the same local rule. In this paper we use the uniform synchronous CLA [4].

3. Cellular learning automata based evolutionary computing

In this section, first some properties of human societies (Table 1), which have directed us to the proposed

Table 1
Similarities between the CLA-EC and human society

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

Initialize *p* to [1/s,1/s, ,1/s] where s is the number of actions
While not done ***do***
  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

Fig. 2. Pseudo-code of a variable-structure learning automaton.

algorithm, are discussed. Every human society has a number of individuals as its basic units. Each individual makes certain choices, adopts certain attitudes, and operates in certain emotional modes. Individuals affect each other within a certain neighborhood mutually and locally, and information about them is local as well. Learning and adaptation are also important features of the individuals in a human society. Each individual tries to learn from his own experiences as well as the experiences of its neighboring individuals. To learn from experiences, there is a simple strategy, in which an individual selects a set of best neighboring individuals with respect to some criteria, and then votes on their experiences. Consequently, the individual can update his belief according to the results of voting and previous self-experience.

The above discussion has motivated us to introduce the CLA-EC. In the CLA-EC, similar to other evolutionary algorithms, the parameters of search space are encoded in the form of genomes. Each genome has two components: the model genome and the string genome. The model genome is a set of LA, which selects a set of actions that determines the string genome. Using a local rule, a reinforcement signal vector is generated for each cell and given to the set of LA 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 the CLA-EC generates a new string genome and compares its fitness with the fitness of its current string genome. If the fitness of the generated genome is better, then the generated string genome becomes the string genome of that cell. This process is repeated until a termination condition is satisfied. The main issues in designing the CLA-EC to solve a problem are defining a suitable genome representation, designing a fitness function, and determining the parameters of the CLA-EC such as the number of cells (population size), the topology of the CLA-EC, and the type of LA for each cell.

Assuming a finite binary search space with the dimension of m, a function optimization problem can be presented as minimization of the real function f:$\{0,1\}^m \rightarrow \Re$, that is, to find

$$\min \{f(\boldsymbol{\xi}) | \boldsymbol{\xi} \in \{0,1\}^m\} \tag{3}$$

In order to use the CLA-EC for optimizing the function f, first a set of LA is assigned to each cell of the CLA-EC. The number of LA assigned to a cell equals the number of bits in the string genome representing a point in the search space of f. Each automaton has two actions called 0 and 1. The probability vectors of all learning automata in all of the cells are initialized to (0.5, 0.5). Then the following steps will be repeated until a termination criterion is met.

I) At the iteration n, every automaton in the cell i chooses one of its actions using its action probability vector.

II) Every cell i generates a new string genome $\omega^i(n)$. The new generated string genome is obtained by concatenating the selected actions of the automata residing in that cell.

Remark. Steps I and II are similar to the situation, in which an individual in a human society makes certain choices and operates in certain emotional modes.

III) Every cell i computes the fitness value of the string genome $\omega^i(n)$. If the computed fitness is better

than the fitness of the current genome, then the new string genome $\omega^i(n)$ becomes the string genome of that cell. That is

$$\boldsymbol{\xi}^i(n)=\begin{cases}\boldsymbol{\xi}^i(n-1) & f(\boldsymbol{\xi}^i(n-1)) \leqslant f(\omega^i(n)) \\ \omega^i(n) & f(\boldsymbol{\xi}^i(n-1)) > f(\omega^i(n))\end{cases} \quad (4)$$

Remark. Step III is similar to the situation, in which an individual in a human society tries to learn from his previous self experience.

IV) Among the neighboring cells of the cell i, *Se* cells are selected. This selection is based on the fitness values of the neighboring cells and a selection strategy such as the truncation strategy.

Remark. In a one-dimensional CLA-EC (Fig. 4), for the neighborhood radius of r, each cell has $2r+1$ neighbors. In simulations, when r is 1 (resp. 2), *Se* is considered to be 1, 2, or 3 (resp. 1, 2, 3, 4, or 5). Choosing $2r+1$ as the value of *Se* simplifies step IV of the CLA-EC. This is because that the algorithm no longer requires a selection strategy; hence, the algorithm has lower time complexity.

V) Based on the selected cells, every cell i generates a reinforcement vector and uses it as an input to its learning automata. Let $H_s(i)$ be the set of the selected neighbors of the cell i. Define $N_{i,j}(k)$, the number of cells in $H_s(i)$ whose jth LA have selected their kth actions, as

$$N_{i,j}(k) = \Sigma_{l \in H_s(i)} \delta(\xi^{l,j}(n) = k) \quad (5)$$

where,

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

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

$$\beta^{i,j}(n)=\begin{cases} u(N_{i,j}(1)-N_{i,j}(0)\,) \text{ if } \xi^{i,j}(n)=0 \\ u(N_{i,j}(0)-N_{i,j}(1)) \text{ if } \xi^{i,j}(n)=1\end{cases} \quad (6)$$

where $u(.)$ is the step function. By Eq. (6), the cell i votes on the experiences of the selected neighboring cells. It means that, for the jth position of the string genome, if the majority of cells in $H_s(i)$ have chosen the same action as (resp. different from) the cell i has, then a reward (resp. penalty) signal is given to the jth learning automaton of the cell i.

VI) All LA update their internal structures by using their learning algorithm and the given reinforcement signals.

Remark. Steps IV, V, and VI are similar to the situation, in which an individual in a human society selects some experts from the people he knows, and then updates his belief according to the result of voting amongst the experiences of the selected people and previous self-experience.

The overall operation of the CLA-EC is summarized in Fig. 3. 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, which is distributed over the search space, and this makes it less susceptible to get trapped in a local optima.

The CLA-EC is not the first evolutionary algorithm, in which the LA have been used. In [11], an algorithm called the StGA has considered the whole population as a single learning automaton. Each genome in the population has been considered as an action of the automaton and the fitness of each genome is replaced by a probability. The evaluation of a genome in the environment results in updating all genomes probabilities. When the crossover operator is applied, the fitness of the offspring is set to the average fitness of two parents. This algorithm is limited to small populations with a short genome length. In [12] an algorithm called the GLA has been proposed. In this algorithm, a corresponding population of probability genomes replaces the population of bit string genomes in the Genetic Algorithm. The value in each position of a genome represents the probability of having the allele value of one in the corresponding position of the bit string. A population of bit strings is stochastically generated in each generation by sampling the distributions of the population. By using each probability string, a single bit string is created in each generation. Then each bit string has its performance evaluated by the test environment like the evaluation in the GA. The fitness of a probability string is equal to that of the corresponding bit string. The probabilities at each position are regarded as the action selection probabilities of a two-action learning automaton. The probabilities are updated in each generation based on a learning algorithm. The mutation operator is not defined, and the crossover operator is defined like the crossover in the GA. In [13], an estimation of distribution algorithm, called the LAEDA, has been presented. In the LAEDA, a set of learning automata is used to model the population of genomes. In each generation, a population of genomes is generated by a 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 learning automata

```
Initialize.
While not done do
    For each cell i in the CLA-EC do in parallel
        Generate a new string genome
        Evaluate the new string genome
        If f(new string genome)> f(old string genome) then
            Accept the new string genome
        End if
        Select Se cells from the neighbors of cell i
        Generate the reinforcement signal vector
        Update LA of cell i
    End parallel for
End while
```

Fig. 3. The pseudo-code of the CLA-EC.

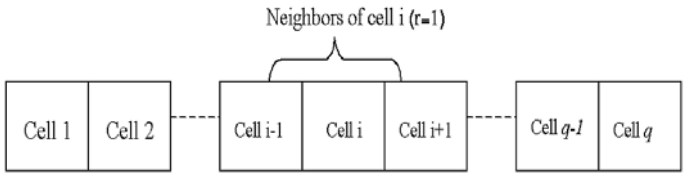


Fig. 4. A one-dimensional (linear) CLA-EC with neighborhood radius one ($r = 1$), a wrap around connection and q cells.

use the reinforcement signal vector to update their internal structures. The process is iterated until a predefined termination condition is met. In this algorithm mutation and crossover operators are not defined.

The CLA-EC differs from the StGA and the GLA models in two respects. 1) Unlike the StGA and the GLA, the CLA-EC model does not use crossover and mutation operators. 2) Unlike the StGA and the GLA, in the CLA-EC each genome only interacts with its predefined neighboring genomes. The CLA-EC is similar to the LAEDA, because both do not have crossover and mutation operators, but the CLA-EC uses different sets of LA to model different genomes, while the LAEDA uses one set of LA to model the whole population. The CLA-EC and the LAEDA both use a subset of the selected genomes from the population to generate a reinforcement signal vector.

The CLA-EC is different from the other fine-grained PEAs reported in literature in two respects: 1) The definition of the local rule and 2) The structure of the individuals. In the other fine-grained PEAs, the local rule is a combination of local selection, crossover, and mutation, but in the CLA-EC the local rule is a combination of the local selection and the reinforcement signal vector generation schema. An individual in fine-grained PEAs is commonly a string genome, while in the CLA-EC an individual is composed of a string genome and a model genome (a set of LA).

4. Simulation results

In this section, we study the performance of the CLA-EC by conducting extensive experiments on two optimization problem classes: real-valued function optimization and data clustering problems. In this regard, we have chosen a CLA-EC model with a linear topology, a wrap around connection, and q cells. If the radius of neighborhood is one, the neighbors of the cell i are the cells $i-1$ and $i+1$ as indicated in Fig. 4. The architecture of a cell has been shown in Fig. 5. Each cell is equipped with m learning automata. The string genome determiner compares the new string genome with the current string genome residing in the cell. The string genome with higher quality becomes 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 a CLA-EC model with a linear topology and neighborhood radiuses of 1 and 2.

4.1. Function optimization

This section presents the simulation results for five function optimization problems and then compares them to the results obtained by using the Simple Genetic Algorithm (SGA) [1], the compact Genetic Algo-

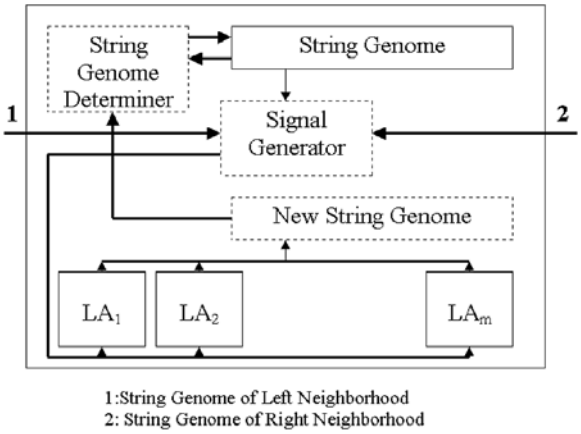


Fig. 5. The structure of a cell.

rithm (cGA) [3], and the Population Based Incremental Learning (PBIL) [15] in terms of solution quality and the number of function evaluations for convergence of the algorithms with a given population size. The CLA-EC completely converges, when all the learning automata residing in all of the cells converge, i.e. the population (the set of string genomes residing in the cells) remains unchanged. Each quantity in the reported results is the average taken over 20 runs. The population size (the number of cells) varies from 3 to 49 with the step of two. The proposed algorithm is tested with L_{RI} and L_{RP} learning algorithms. For the sake of convenience in presentation, we use the notation *CLA-EC* (*automata*(a, b), r, *Se*, q) for referring to a CLA-EC model with q cells, neighborhood radius of r, the number of selected cells *Se*, and learning automata with reward and penalty parameters of a and b respectively. The algorithm terminates, when all learning automata converge. The proposed algorithm is tested with five well-known function minimization problems, which have been used in [7].

1) The first De Jong function (sphere)

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

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

2) The second De Jong function (Rosenbrock's saddle)

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

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

3) The third De Jong function (step)

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

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

4) The fourth De Jong function (Quadratic function)

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

This function has been designed to test the behavior of minimization algorithms in the presence of noise.

5) The fifth De Jong function (Shekel's Foxholes)

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

To study the convergence speed of the CLA-EC, we report the best, the worst, the mean, and the standard deviation of fitness of all cells for each function. Figures 6 and 7 show the simulation results 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 the CLA-EC converges to a near optimal solution.

The size of the CLA-EC (population size) is another important parameter which affects the performance of the CLA-EC. Figure 8 shows how this parameter affects the convergence speeds 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 in iteration. Simulation results show that as the size of the CLA-EC increases, the convergence speed increases. However, it is observed that in some experiments, if the size of the CLA-EC increases beyond a value, no increase in the performance happens.

To study the effects of the neighborhood radius and *Se* on the performance of the CLA-EC, several simulations have been conducted with the neighborhood radiuses of one and two. Most of simulations show that the CLA-EC with the radius of one produces high quality solutions. For F_1, F_2, and F_4 when *Se* is set to 2 or 4, the CLA-EC performs better in terms of convergence speed and the quality of the obtained solutions. For F_3, when *Se* is set to 2 or 4, the CLA-EC converges faster, but the obtained solutions are not optimal. When *Se* is

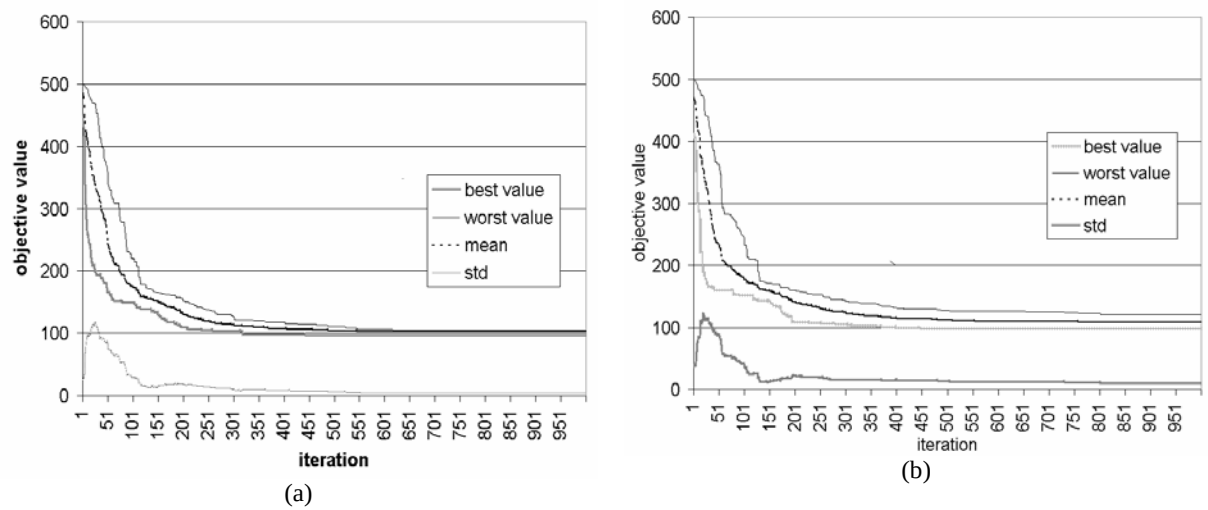


Fig. 6. Function F_5. (a) *CLA-EC* (L_{RP} (0.01, 0.01), 1, 2, 5); (b) *CLA-EC*(L_{RI} (0.01, 0.01), 1, 2, 5).

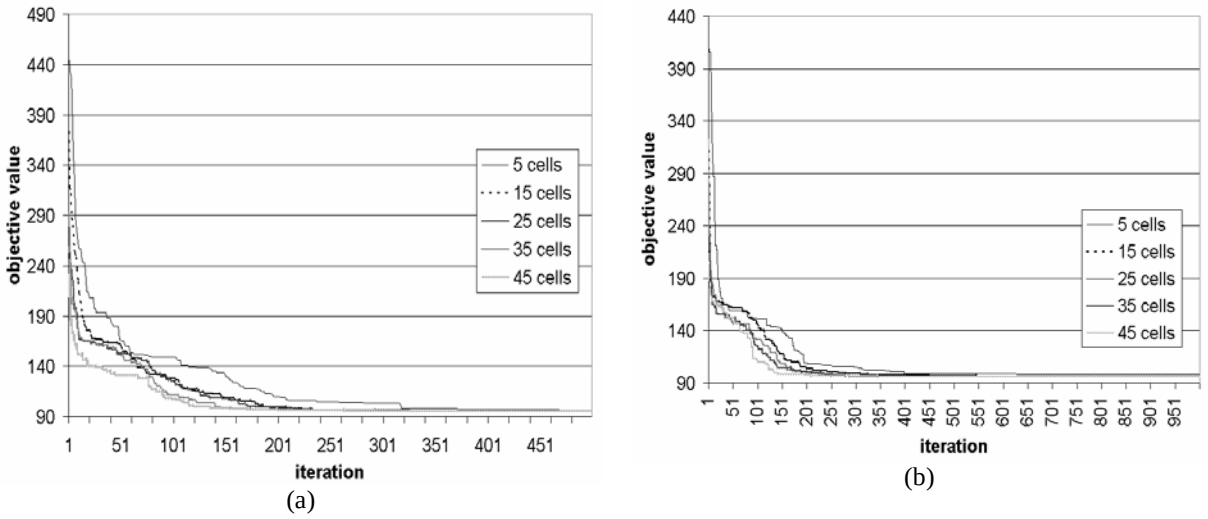
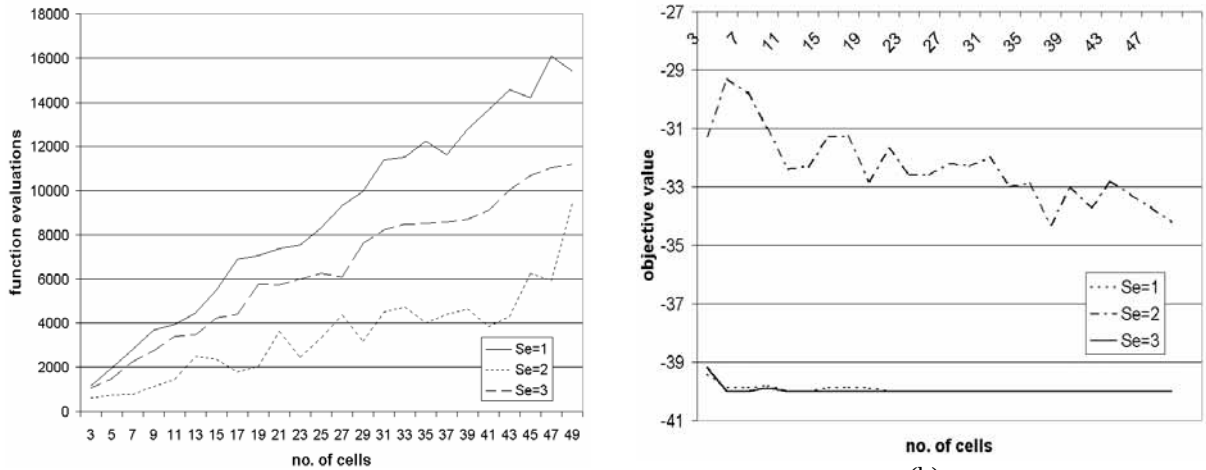


Fig. 7. The best value obtained at each iteration for function F_5. (a) *CLA-EC*(L_{RP} (0.01, 0.01), 1, 2, $-$); (b) *CLA-EC*(L_{RI} (0.01, 0.01), 1, 2, $-$).

set to 1, 3, or 5, the CLA-EC converges to the optimal solution in all of the runs. For F_5, when *Se* is set to 3 or 5, the convergence rate of the CLA-EC increases.

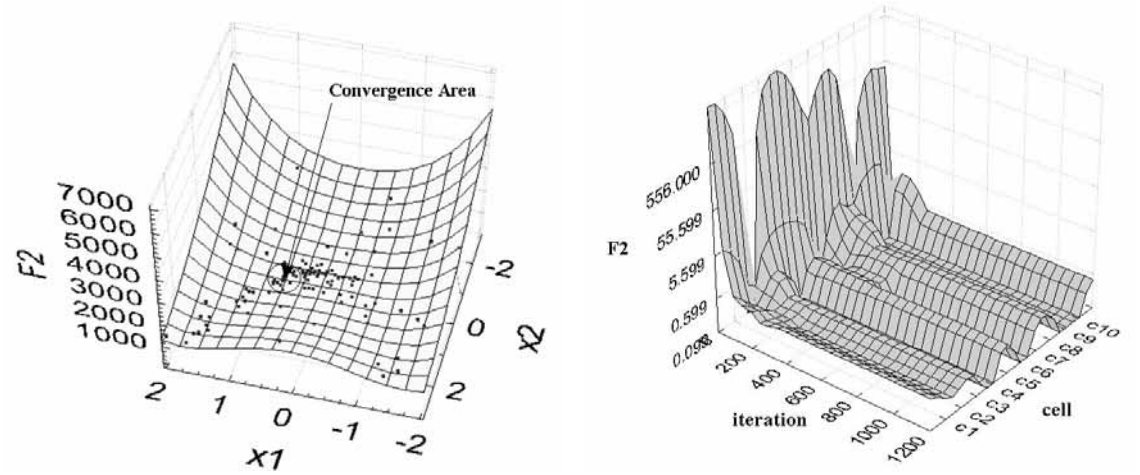
Figure 9(a) shows the evolution of the string genomes of *CLA-EC*(L_{RP}(0.01, 0.01), 1, 2, 10) for F_2 during the optimization process. At the beginning of the algorithm, the genome diversity of the CLA-EC is high and it decreases as the CLA-EC approaches the solution. After the CLA-EC converges, all the genomes in the CLA-EC will be in the convergence area as shown in Fig. 9(a). That is, all string genomes are either optimal or near optimal solutions of F_2. Figure 9(b) shows the fitness evolution of all genomes of *CLA-EC*(L_{RP}(0.01, 0.01), 1, 2, 10) during a typical run. This figure indicates that the fitness of genomes in the CLA-EC gets closer to each other as the optimization process proceeds.

To show the performance of the CLA-EC for the real-valued function optimization, we compare it to the SGA, the cGA [3], and the PBIL [15]. The SGA uses the two-tournament selection without replacement and uniform crossover with the exchange probability of 0.5. Mutation is not used, and crossover is applied with the probability of one. The PBIL uses the learning rate of 0.01 and the number of the selected genomes is half of the population size. The parameters of the cGA are the



(a) (b)

Fig. 8. Function F_3. a) *CLA-EC*(L_{RP} (0.01, 0.01), 1, −, −); b) *CLA-EC*(L_{RP} (0.01, 0.01), 2, −, −).



(a) (b)

Fig. 9. a) The genome dispersion of *CLA-EC*(L_{RP} (0.01, 0.01), 1, 2, 10) for F_2; b) The fitness evolution of all genomes of *CLA-EC* (L_{RP} (0.01, 0.01), 1, 2, 10) during a typical run for F_2.

same as those used in [3]. Convergence is considered as the termination condition for all of the algorithms. For the CLA-EC, *Se* is set to 2 for F_1, F_2, F_4, and F_5, and it is set to 3 for F_3. The results have been reported in Figures 10–14 indicating the superiority of the proposed algorithm over the SGA, the PBIL, and the cGA.

4.2. Data clustering

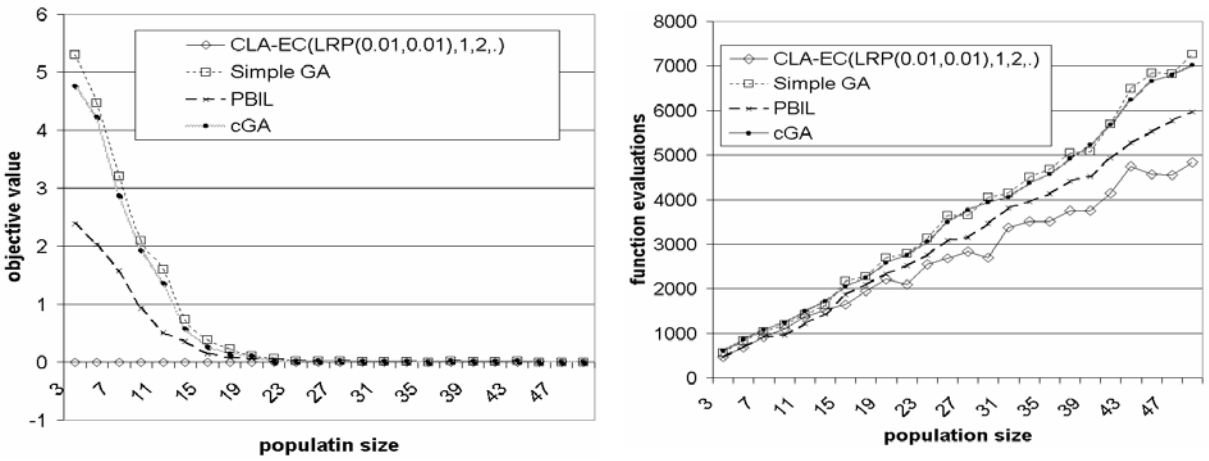
In a clustering problem [8], a data set $S = \{\mathbf{x}_1, \ldots, \mathbf{x}_M\}$ is given in N-dimensional Euclidean space, where $\mathbf{x}_i \in R^N$ and M is the number of data. Considering K clusters, represented by $C_1, \ldots, C_K$, the clusters should satisfy the following conditions,

$$C_i \neq \phi, i = 1, \ldots, K,$$

$$\bigcup_{i=1}^{K} C_i = S,$$

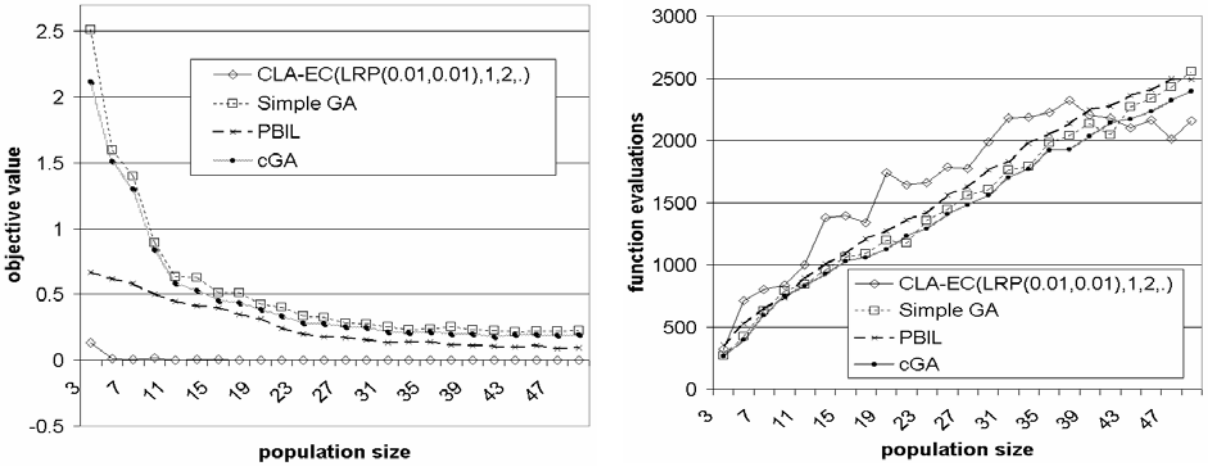
$$C_i \cap C_j = \phi, i \neq j, i, j = 1, \ldots, K.$$

We propose using the CLA-EC to determine K cluster centers for clustering the date set $S = \{\mathbf{x}_1, \ldots, \mathbf{x}_M\}$. The sum of squared distances of the data from their respective cluster centers is taken as the clustering metric and denoted by f. The aim is to minimize f by



(a) (b)

Fig. 10. The *CLA-EC*(L_{RP} (0.01, 0.01), 1, 2, $-$), the SGA, the PBIL, and the cGA for function F_1. a) Objective value; b) Function evaluations.



(a) (b)

Fig. 11. The *CLA-EC*(L_{RP} (0.01, 0.01), 1, 2, $-$), the SGA, the PBIL, and the cGA for function F_2. a) Objective value; b) Function evaluations.

finding the appropriate cluster centers. The proposed algorithm consists of three phases: the preprocessing phase, the CLA-EC phase, and the clustering phase.

The preprocessing phase: The aim of the preprocessing phase is to reduce the size of the search space. In this regard, at first the largest and the smallest values of each dimension of the data set are computed as follows:

$$\min_j = \min_{1 \leqslant i \leqslant M} \{\mathbf{x}_{ij}\}$$

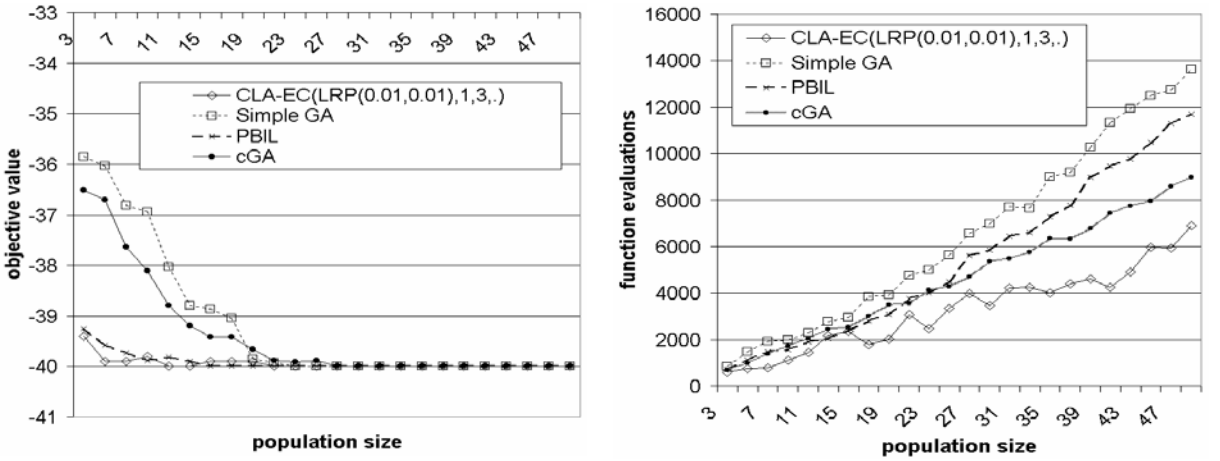
$$\max_j = \max_{1 \leqslant i \leqslant M} \{\mathbf{x}_{ij}\}$$

$$\Delta_j = \max_j - \min_j$$

where $\mathbf{x}_{ij}$ is the jth component of $\mathbf{x}_i$. Then we define $R' = [0, \Delta_1] \times \ldots \times [0, \Delta_N]$.

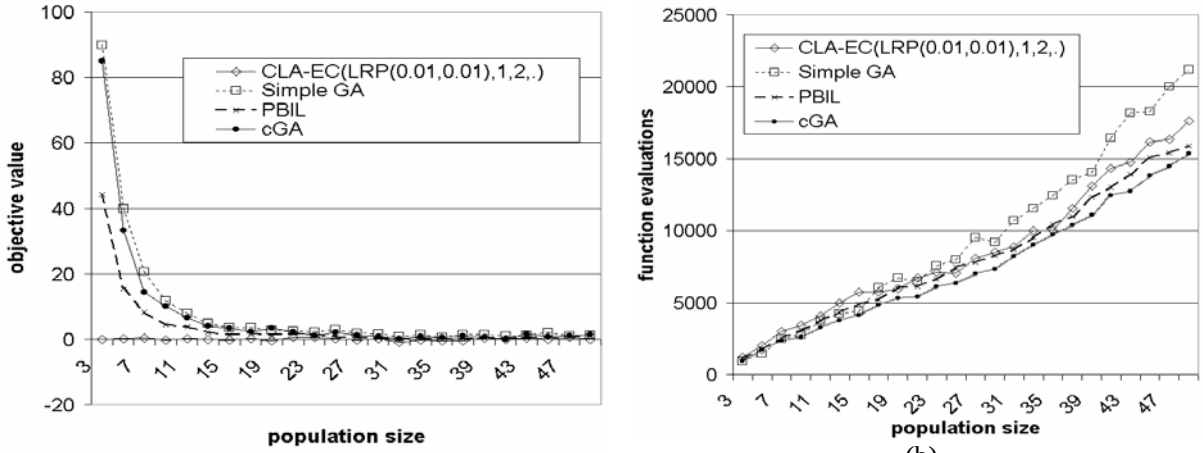
The CLA-EC phase: In the CLA-EC phase, the clusters are optimized with respect to the squared error criterion. The characteristics of the chosen CLA-EC are as follows.

String genome representation: Each string genome is represented by a binary string consisting of $K \times N$ parts. The $(i \times N + j)$th part is a sequence, which contains the corresponding encoded real number for the jth dimension of the cluster center i in R'. Let λ'_{ij} be the $(i \times N + j)$th part of the string genome. If the binary representation of λ'_{ij} has w_{ij} bits, then in an N-dimensional space with K cluster centers, the string



(a) (b)

Fig. 12. The *CLA-EC*(L_{RP} (0.01, 0.01), 1, 3, $-$), the SGA, the PBIL, and the cGA for function F_3. a) Objective value; b) Function evaluations.



(a) (b)

Fig. 13. The *CLA-EC*(L_{RP} (0.01, 0.01), 1, 2, $-$), the SGA, the PBIL, and the cGA for function F_4. a) Objective value; b) Function evaluation.

genome length is $m = \Sigma\Sigma w_{ij}$ bits, where $1 \leqslant i \leqslant K$ and $1 \leqslant j \leqslant N$.

Fitness function: To compute the fitness of ξ, first we compute each λ'$_{ij}$ by decoding the $(i \times N + j)$th part of ξ, and then we set λ_{ij} to be $(\lambda'_{ij} + \min_j)$. Finally the fitness of a genome is computed as follows:

$$f(\xi) = f(\boldsymbol{\lambda}_1, \ldots, \boldsymbol{\lambda}_K) = \sum_{i=1}^{M} (A_i^*)^2,$$

where,

$$A_i^* = \min_{1 \leqslant j \leqslant K} A_{ij}; A_{ij} = \|\mathbf{x}_i - \boldsymbol{\lambda}_j\|;$$

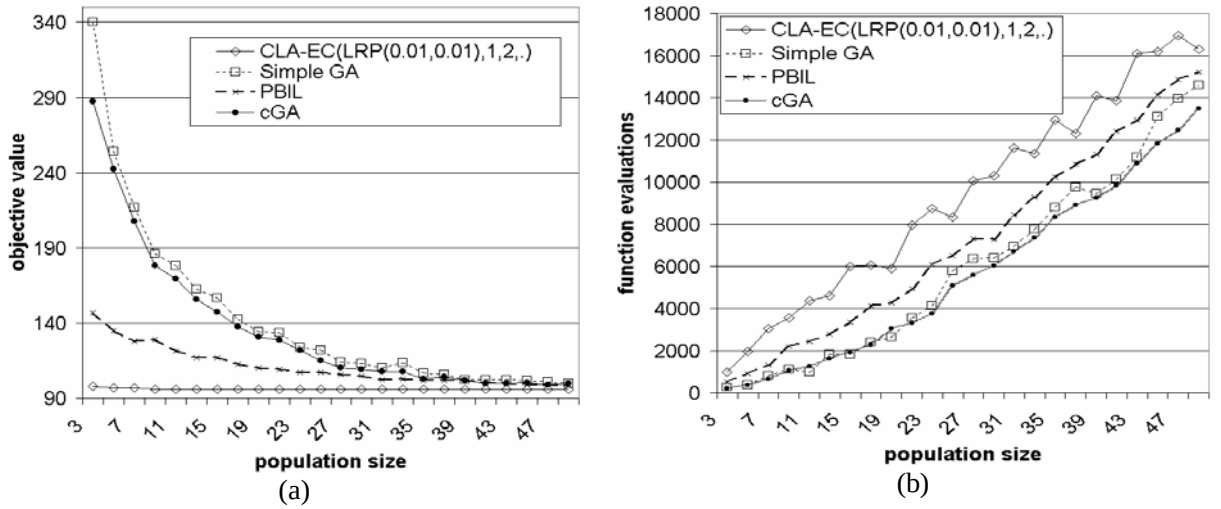
$$\boldsymbol{\lambda}_j = (\lambda_{j1}, \ldots, \lambda_{jN})$$

Termination Criteria: The CLA-EC stops after a pre-determined number of iterations and the best string genome in the last iteration is the solution of the clustering problem. For the following experiments, the maximum number of iterations is set to 200.

The Clustering Phase: In this phase, the clusters are created based on the clustering centers encoded in the best string genome of the pervious phase. This is done by assigning each point $\mathbf{x}_i$, $i = 1, \ldots, M$, to one of the clusters C_k with the center $\boldsymbol{\lambda}_k$ such that,

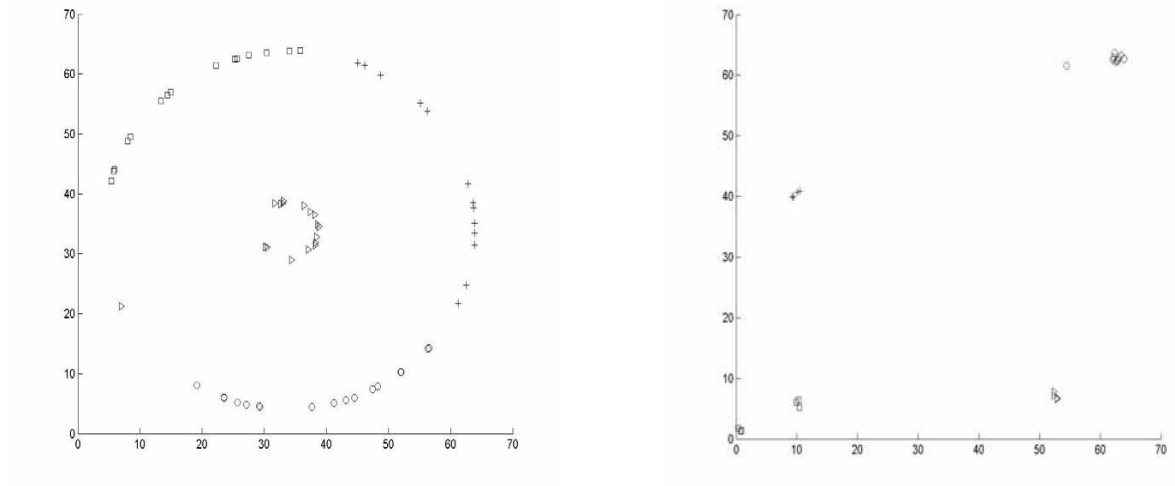
$$C_k = \arg\min_{1 \leqslant j \leqslant K} A_{ij},$$

where



(a) (b)

Fig. 14. The *CLA-EC*(L_{RP} (0.01, 0.01), 1, 2, $-$), the SGA, the PBIL, and the cGA for function F_5. a) Objective value; b) Function evaluations.



(a) (b)

Fig. 15. (a) Data 1; (b) Data 2.

$$A_{ij} = \|\mathbf{x}_i - \boldsymbol{\lambda}_j\|.$$

All ties are resolved arbitrary.

Several simulations are conducted in order to evaluate the efficiency of the proposed method. Then the results are compared to the results obtained by using the K-means algorithm. Simulations are conducted with nine different data sets denoted by Data 1, Data 2, Data 3, Data 4, Data 5, Data 6, Data 7, Data 8, and IRIS data sets. The characteristics of these data sets have been given below.

Data 1: A two-dimensional data set with 4 clusters and 59 points as shown in Fig. 15(a).

Data 2: A two-dimensional data set with 4 clusters and 25 points as shown in Fig. 15(b).

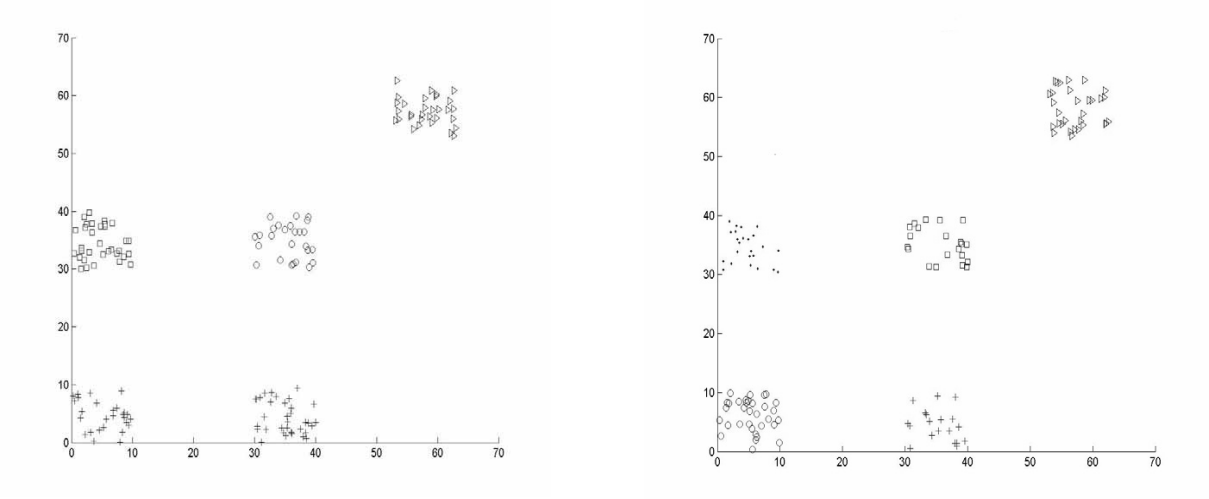
Data 3: A two-dimensional data set with 4 clusters and 170 points as shown in Fig. 16(a).

Data 4: A two-dimensional data set with 5 clusters and 128 points as shown in Fig. 16(b).

Data 5: A two-dimensional data set with 5 clusters and 100 points as shown in Fig. 17(a).

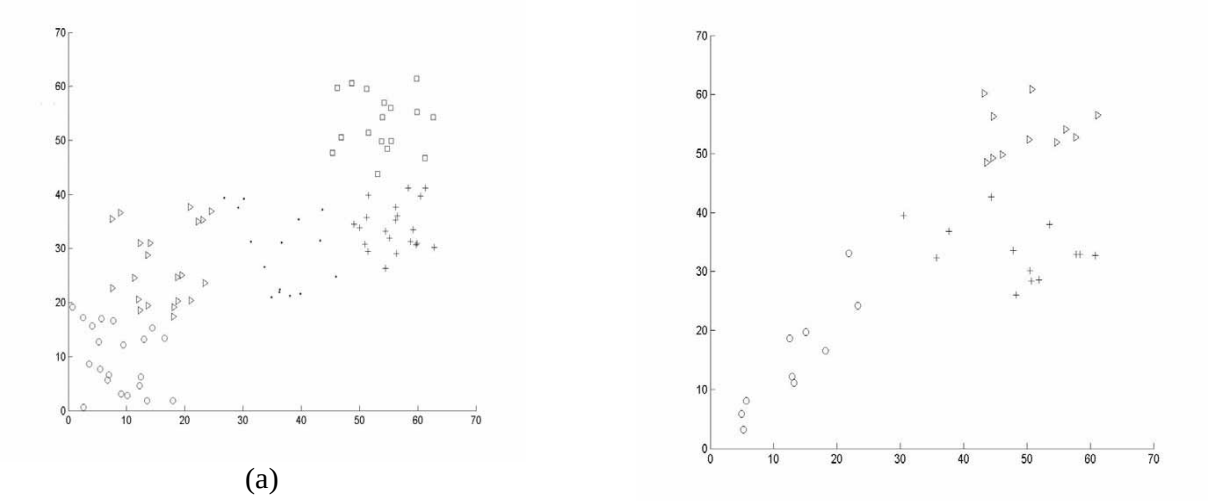
Data 6: A two-dimensional data set with 3 clusters and 35 points as shown in Fig. 17(b).

Data 7: A two-dimensional data set with 2 clusters and 131 points as shown in Fig. 18(a).



(a) (b)

Fig. 16. (a) Data 3; (b) Data 4.



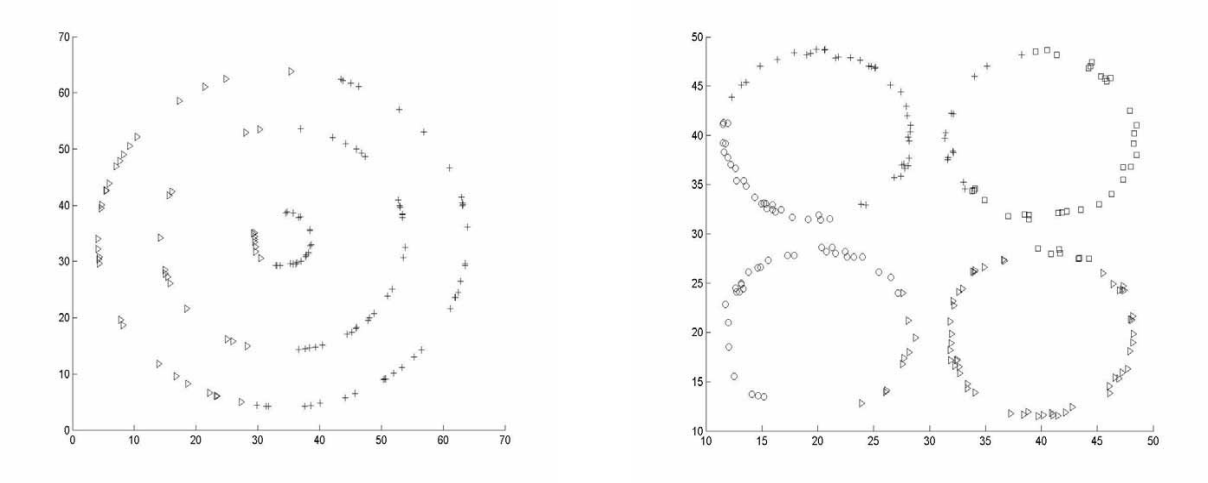
(a) (b)

Fig. 17. (a) Data 5; (b) Data 6.

Data 8: A two-dimensional data set with 4 clusters and 204 points as shown in Fig. 18(b).

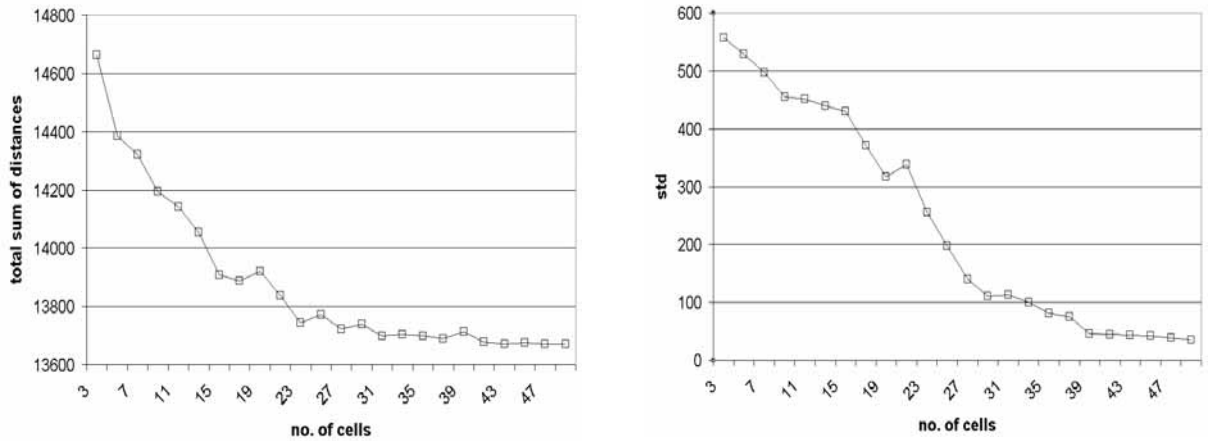
IRIS: This data set represents different categories of irises with four feature values. These feature values represent the sepal length, sepal width, petal length, and the petal width in centimeters. This data set has 3 clusters with 150 points.

We conduct extensive simulations to show that how different parameters of the CLA-EC affect the performance of clustering. For each simulation, the maximum number of iterations is taken to be 200. By careful inspection of the results, it is clear that as the number of cells increases, the mean and the standard deviation of the results decrease. Also, it has been found that better results are obtained, when each automaton uses L_{RP} or $L_{R\varepsilon P}$ learning algorithms with *Se*=1. Figures 19 and 20 show the effect of the number of cells on clustering the data sets Data 8 and IRIS by using *CLA-EC*(L_{RP}(0.01, 0.01), 1, 1,−). It has been shown that as the number of cells increases, the quality of clustering improves. Figure 21 shows the fitness evolution of all genomes of *CLA-EC*(L_{RP}(0.01, 0.01), 1, 1, 15)



(a) (b)

Fig. 18. (a) Data 7; (b) Data 8.



(a) (b)

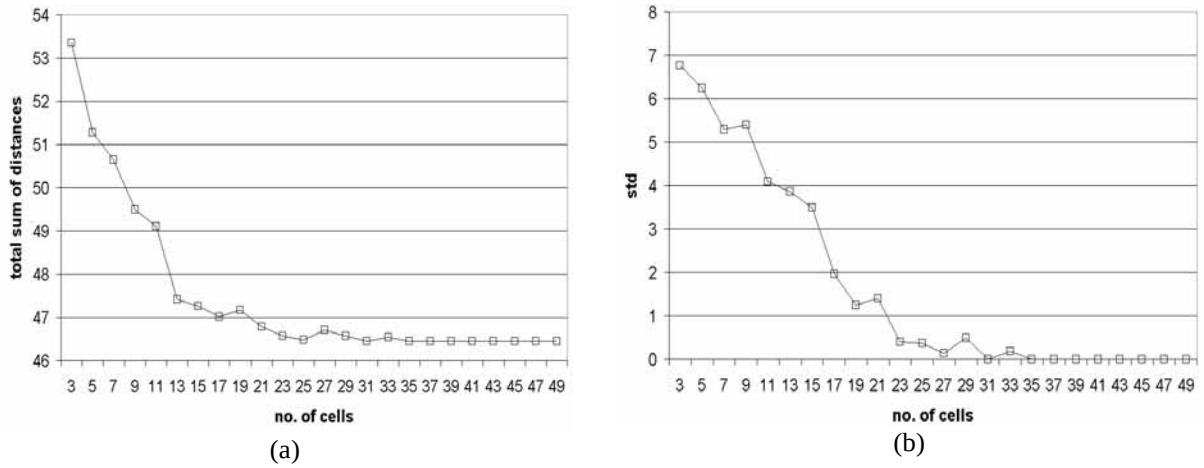
Fig. 19. The effect of the number of cells on the quality of clustering for Data 8 using CLA-EC(L_{RP}(0.01, 0.01), 1, 1, −). (a) The mean and (b) the standard deviation over 50 runs.

during a typical run with Data 8. These figures indicate that the fitness values of all genomes in the CLA-EC approach a value, which is a near optimal solution.

In the following experiment, we compare the results of the proposed algorithm with those of the K-means algorithm [20]. In this experiment, the CLA-EC has 5 cells, each learning automaton uses L_{RP} learning algorithm with $a = b =$0.1, *Se* is 1, and the maximum number of iterations is set to 200. In this section, each quantity is the average taken over 50 runs. For Data 1, it has been found that the CLA-EC provides the optimal value of 9502.44 in 28% of the runs, whereas the K-means algorithm attains this value in 8% of the runs. Both algorithms get trapped at a local minimum in the other runs. For Data 2, the CLA-EC attains the best value of 239.10 in all of the runs. The K-means algorithm, on the other hand, attains this value in 28% of the runs and in the other runs it gets stuck at some local minima (such as 3433.77, 3497.16, or 5551.52). For Data 3, Data 4, Data 5, Data 6, Data 7, Data 8, and IRIS data sets, the CLA-EC 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%, and 30% of the runs, respectively. The best values attained by the K-means for these data sets are 15545.09, 1873.71, 5515.34, 3000.43, 44100.3,

Table 2
The results of the *CLA-EC*(L_{RP} (0.1, 0.1), 1, 1, 5) (maximum 200 iterations) and the K-means algorithm for Data 1, 2, 3, 4, 5, 6, 7, 8, and IRIS data sets

Data Set	(CLA-EC) Mean	(CLA-EC) 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	5920.90	817.05
6	3078.00	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



(a) (b)

Fig. 20. Effect of the number of cells on the quality of clustering for IRIS data set using CLA-EC(L_{RP}(0.01, 0.01), 1, 1, −). (a) The mean and (b) the standard deviation over 50 runs.

13670.63, and 46.44 in 20%, 30%, 2%, 30%, 26%, 14%, and 24% of the runs, respectively.

Table 2 has summarized the results of the experiments. The columns 'Mean' and 'Std' show the mean and standard deviation over 50 runs respectively. 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 algorithm for Data 1, Data 2, Data 3, Data 4, Data 5, Data 6, Data 7, and IRIS data sets. For Data 8, it has been found that the number of cells required by the CLA-EC to outperform the K-means algorithm should be greater than 15 and the *CLA-EC*(L_{RP}(0.1, 0.1), 1, 1, 15) attains the best value of 13670.63 in 16% of the runs.

5. Conclusion

In this paper a new approach of fine-grained PEAs, called the CLA-EC, has been proposed to overcome the existing uncertainty in the other fine-grained PEAs. In the CLA-EC, each genome in the population is assigned to a cell of the CLA and each cell in the CLA is equipped with a set of LA. The set of actions selected by the set of LA 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. Based on the received signal, each learning automaton 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. This work can be extended in several aspects. In this paper, we have used the simplest variable learning automata with linear learning algorithms. However, one can use other learning schema such as nonlinear learning algorithms and even fixed structured learning automata instead of variable structure learning automata. Furthermore, we have used a linear CLA-EC in our studies, and the effect of the topological properties of the CLA-EC is an exciting topic for future research.

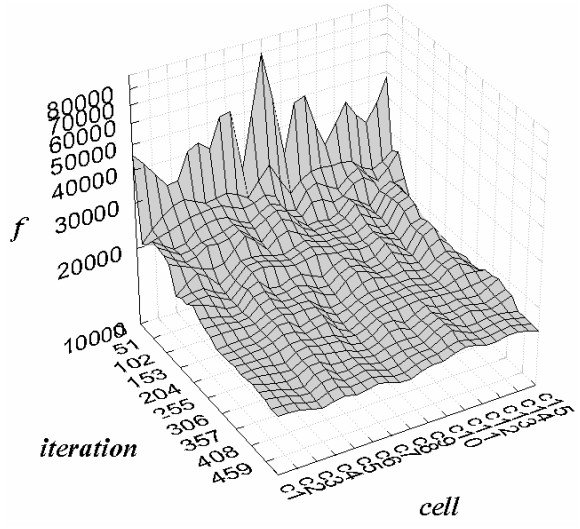


Fig. 21. The fitness evolution of all genomes of *CLA-EC*(L_{RP} (0.01, 0.01), 1, 1, 15) in a typical run with Data 8.

Acknowledgment

The authors are pleased to acknowledge the constructive comments of the anonymous reviewers.

References

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

[2] D. Whitley, A Review of Models for Simple Genetic Algorithms and Cellular Genetic Algorithms, in: *Applications of Modern Heuristic Methods*, V. Rayward-Smith, editor, Alfred Waller, 1995, pp. 55–67.

[3] G.R. Harik, F.G. Lobo and D.E. Goldberg, The compact genetic algorithm, *IEEE Transactions on Evolutionary Computing* **3**(4) (1999), 287–297.

[4] H. Beigy and M.R. Meybodi, A mathematical framework for cellular learning automata, *Advances in Complex Systems* **7**(3,4) (September 2004), 295–319,

[5] H. Beigy and M.R. Meybodi, *A Self-Organizing Channel Assignment Algorithm: A Cellular Leaning Automata Approach*, (Vol. 2690) of Springer-Verlag Lecture Notes in Computer Science, Springer-Verlag, 2003, 119–126.

[6] H. Beigy and M.R. Meybodi, Open synchronous cellular learning automata, *Journal of Computer Science and Engineering* **1**(4b) (2003), 39–51.

[7] K.A. DeJong, *The Analysis of the Behavior of a Class of Genetic Adaptive Systems*, Ph.D. Dissertation, University of Michigan, Ann Arbor, 1975.

[8] K. Fukunaga, *Introduction to Statistical Pattern Recognition*, Academic Press, New York, 1990.

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

[10] M.A.L. Thathachar and P.S. Sastry, Varieties of Learning Automata: An Overview, *IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics* **32**(6) (2002), 711–722.

[11] M. Munetomi, Y. Takai and Y. Sato, StGA: an application of genetic algorithm to stochastic learning automata, *Syst. Comput. Jpn.* **27** (1996), 68–78.

[12] M.N. Howell, T.J. Gordon and F.V. Brandao, Genetic Learning Automata for Function Optimization, *IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics* **32**(6) (2002), 804–815.

[13] R. Rastegar and M.R. Meybodi, *A New Estimation of Distribution Algorithm based on Learning Automata*, in the Proceedings of the 12^{th} IEEE Congress on Evolutionary Computation 2005, UK, 2005, 1982–1986.

[14] S. Baluja, A Massively Distributed Parallel Genetic Algorithm, Technical Report No. CMU-CS-92-196R, Carnegie Mellon University, Pittsburgh, PA, 1992.

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

[16] S. Baluja, *Structure and Performance of Fine-Grain Parallelism in Genetic Search*, Proceedings of the Fifth International Conference on Genetic Algorithms, Morgan Kaufmann (San Mateo, CA), 1993, 155–162.

[17] S. Ghanbari and M.R. Meybodi, *Learning Automata Based Algorithms for Mapping of a Class of Independent Tasks over Highly Heterogeneous Grids*, (Vol. 3470) of Springer-Verlag Lecture Notes in Computer Science, Springer Verlag (Berlin), 2005, 681–690.

[18] S.M. Gorges, *Explicit Parallelism of genetic Algorithms through Population Structures*, Parallel Problem Solving from Nature, Springer-Verlag (Berlin), 1991, 150–159.

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

[20] S.Z. Selim and M.A. Ismail, K-means-type algorithm: generalized convergence theorem and characterization of local optimality, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **6**(1) (1984), 81–87.