New Generation Computing
https://doi.org/10.1007/s00354-022-00159-1

A New Evolutionary Model Based on Cellular Learning Automata and Chaos Theory

Bagher Zarei[1] · **Mohammad Reza Meybodi**[2] · **Behrooz Masoumi**[3]

Received: 20 June 2020 / Accepted: 6 February 2022

Abstract
In this paper, a new fine-grained evolutionary model, called CCLA-EM, is proposed for solving the optimization problems, which greatly overcomes the premature convergence problem of the existing evolutionary algorithms. In the proposed model, a combination of an evolutionary algorithm with a cellular learning automaton is used. The population individuals are distributed on the cells of a cellular learning automaton. Each individual interacts and cooperates with the individuals of neighboring cells to reach the global optimum. Distributing the population individuals on the cells of a cellular learning automaton allows the parallel implementation of the proposed model. Also, in different stages of the proposed model, numbers generated by a chaotic process are used instead of random ones. The use of numbers generated by a chaotic process leads to a complete search of the search space and hence avoids being trapped in local optima. Experiments on various benchmarks of the community structure detection problem indicate the superiority of the proposed model to the well-known algorithms GA-net and ICLA-net.

Keywords Evolutionary algorithm · Cellular evolutionary algorithm · Cellular automata · Learning automata · Cellular learning automata · Chaos theory · Community structure detection

✉ Bagher Zarei
zarei.bager@yahoo.com

[1] Faculty of Computer and Information Technology Engineering, Shabestar Branch, Islamic Azad University, Shabestar, Iran

[2] Department of Computer Engineering and Information Technology, Amirkabir University of Technology, Tehran, Iran

[3] Faculty of Computer and Information Technology Engineering, Qazvin Branch, Islamic Azad University, Qazvin, Iran

Published online: 21 February 2022

Introduction

In applied mathematics and computer science, an optimization problem is the problem of finding the best solution from all feasible solutions. Optimization problems can be classified into two categories depending on whether the variables are *continuous* or *discrete*. Most of the interesting real-world optimization problems are NP-hard. It is widely believed—though not yet proven—that NP-hard problems are intractable, which means that there is no efficient algorithm (i.e., one that scales polynomially) that is guaranteed to find an optimal solution for such problems. They require exponential time to be solved. Metaheuristics constitute an important alternative to solve this class of problems. Evolutionary algorithms (EAs) such as genetic algorithm (GA) and particle swarm optimization algorithm (PSO) are a class of metaheuristics for solving the NP-hard optimization problems [1–5].

In most EAs like GA, individuals (parents) to perform evolutionary or variation operators are selected from the whole population. So, the global fitness distribution of the population must be determined. However, in nature, a global selection does not exist, and the global fitness distribution cannot be determined either. The real natural selection only occurs in a local environment, and each individual can only interact with those around it. That is, in some phases, the natural evolution is just a kind of local phenomenon. The information can be shared globally only after a process of diffusion [6–8]. Furthermore, most EAs have the problem of being trapped in a local optimum, and hence premature convergence [9]. In this paper, a new fine-grained evolutionary model, called CCLA-EM, is proposed, which greatly resolves the mentioned problems. In the proposed model, a combination of an evolutionary algorithm with a cellular learning automaton is used. In the proposed model, each individual of the population is mapped to a cell of a cellular learning automaton. Moreover, n learning automata are assigned to each cell. The variable n is the number of optimization problem variables. The chosen actions by the learning automata residing in a particular cell determine the state (value of the optimization problem variables) of that cell. In the proposed model, the learning automata residing in a particular cell only interact and cooperate with the learning automata of neighboring cells. There is no global selection at all, so global fitness distribution is not required. An individual interacts with its neighbors so that information can be transferred to them. In such a manner, the information is gradually diffused to the whole cellular lattice. The evolutionary mechanism based on the cellular lattice used in the CCLA-EM is closer to the real evolutionary mechanism in nature than the model of the population in traditional EAs. Also, in different stages of the proposed model, chaotic numbers are used instead of random ones. Chaotic numbers are a sequence of numbers generated by a chaotic process. In this paper, for the sake of simplicity, the term "chaotic numbers" is used instead of "a sequence of numbers generated by a chaotic process". It has been shown that using chaotic numbers instead of random ones leads to a significant improvement in the convergence and performance of an EA [9–11].

The rest of this paper is organized as follows: In Sects. 2 and 3, cellular learning automata theory and chaos theory are explained, respectively. Distributed

evolutionary algorithms are described in Sect. 4. In Sect. 5, the proposed evolutionary model is presented. The application of the proposed model on the community structure detection problem and its performance evaluation are given in Sect. 6. Finally, Sect. 7 contains the conclusions of the paper.

Theory of Cellular Learning Automata

In this section, at first cellular automata and then learning automata are explained. Finally, the theory of cellular learning automata, which is a combination of cellular automata and learning automata, is described.

Cellular Automata

Cellular automata (CAs) [12] are mathematical models for systems that consist of large numbers of simple identical components with local interactions. CAs are nonlinear dynamical systems in which space and time are discrete. They are called *cellular* because they are made up of cells like points in a lattice or like squares of checkerboards, and they are called *automata* because they follow a simple rule. The simple components act together to produce complicated patterns of behavior. CAs perform complex computations with a high degree of efficiency and robustness. They are especially suitable for modeling natural systems that can be described as massive collections of simple objects interacting locally with each other. Informally, a d-dimensional CA consists of an infinite d-dimensional lattice of identical cells. Each cell can assume a state from a finite set of states. The cells update their states synchronously on discrete steps according to a local rule. The new state of each cell depends on the previous state of a set of cells including the cell itself that constitutes its neighborhood. The state of all cells in the lattice is described by a configuration. The local rule and the initial configuration of the CA specify the evolution of CA. The local rule tells how each configuration is changed in one step. Formally, a d-dimensional CA can be defined as quadruple $\left(Z^d, \Phi, N, F\right)$, where:

(1) Z^d is a lattice of d-tuples of integer numbers. Each cell in the d-dimensional lattice, Z^d, is represented by a d-tuple $\left(z_1, z_2, \ldots, z_d\right)$.
(2) $\Phi = \left\{\Phi_1, \Phi_2, \ldots, \Phi_s\right\}$ is a finite set of states.
(3) $N = \left(n_1, n_2, \ldots, n_h\right)$ is a finite subset of Z^d called the neighborhood vector, where $n_i \in Z^d$. The neighborhood vector determines the relative position of the neighboring lattice cells from any given cell u in the lattice Z^d. The neighbors of a particular cell u are the set of cells $\left\{u + n_i | i = 1, 2, \ldots, h\right\}$. We assume that there exists a neighborhood function $\mathrm{NF}(u)$ that maps a cell u to the set of its neighbors, that is $\mathrm{NF}(u) = \left\{u + n_1, u + n_2, \ldots, u + n_h\right\}$. For the sake of simplicity, we assume that the first element of the neighborhood vector (i.e., n_1) is equal to d-tuple $(0, 0, \ldots, 0)$ or equivalently $u + n_1 = u$. The neighborhood function $\mathrm{NF}(u)$ must satisfy the following two conditions: (1) $u \in \mathrm{NF}(u), \forall u \in Z^d$ and (2) $u_1 \in \mathrm{NF}\left(u_2\right) \Leftrightarrow u_2 \in \mathrm{NF}\left(u_1\right), \forall u_1, u_2 \in Z^d$. For example, the neighborhood

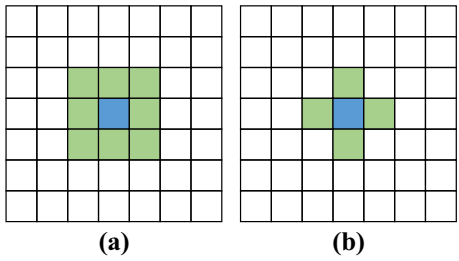


Fig. 1 Neighborhood in the 2-dimensional cellular automaton. **a** von-Neumann neighborhood and **b** Moore neighborhood. The neighborhood radius is considered one

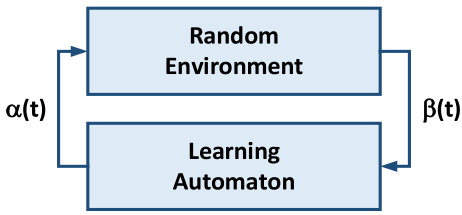


Fig. 2 The interaction of a learning automaton and its environment

vector $N = ((0,0),(-1,0),(1,0),(0,-1),(0,1))$ as shown in Fig. 1a is called the von-Neuman neighborhood.

(4) $F = \Phi^{|N|} \rightarrow \Phi$ is the local rule of the CA. It computes the new state for each cell from the current state of its neighbors.

Learning Automata

The automata approach to learning can be considered as the determination of an optimal action from a set of actions. A learning automaton (LA) can be regarded as an abstract object with a finite number of actions. It selects an action from its finite set of actions and applies it to an environment. The environment evaluates the applied action and sends a reinforcement signal to the LA, as shown in Fig. 2. The reinforcement signal provided by the environment is used to update the internal state of the LA. By continuing this process, the LA gradually learns to select the optimal action, which leads to favorable responses from the environment [13].

Formally, an environment can be defined as triple (α, c, β), where $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$ is a set of inputs, $\beta = \{\beta_1, \beta_2, \ldots, \beta_q\}$ is a set of outputs, and $c = \{c_1, c_2, \ldots, c_r\}$ is a set of penalty probabilities, where each element c_i of c corresponds to one input action α_i. The elements of c characterize the environment. Based on the nature of β, environments could be classified into three categories: P-, Q-, and S-models. The output of a P-model environment has two elements of success or failure. Usually, in P-model environments, a failure (or unfavorable response) is denoted by 1, while a success (or a favorable response) is denoted by 0. In Q-model environments, β can take a finite number of values in the interval [0, 1], while in S-model environments, β lies in the interval [0, 1]. In P-model environments, c_i represents the probability that the application of an action α_i to the environment will result in a penalty output, i.e., $c_i = \Pr\{\beta(t) = 1 | \alpha(t) = \alpha_i\}$ (t denotes time) [13].

Formally, an automaton can be defined as quintuple $(\Phi, \alpha, \beta, F, G)$, where $\Phi = \{\Phi_1, \Phi_2, \ldots, \Phi_s\}$ is a set of states,$\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$ is a set of actions (outputs) that it must choose from, $\beta = \{\beta_1, \beta_2, \ldots, \beta_q\}$ is a set of inputs, $F : \Phi \times \beta \to \Phi$ is the state transition function that defines the transition of the state of the automaton upon receiving an input from the environment, and $G : \Phi \to \alpha$ is the output function that determines the action taken by the automaton. The chosen action at the instant t, denoted by $\alpha(t)$, serves as the input to the environment. The environment evaluates the chosen action according to a fixed unknown probability distribution and outputs a stochastic response, $\beta(t)$, at the instant t, which is considered as the response of the environment to the automaton. Based on the response $\beta(t)$, the state of the automaton, $\Phi(t)$, is updated, and a new action is chosen at the instant $(t + 1)$. An automaton acting in an unknown random environment in this manner, to improve its performance in some specified sense, is referred to as a learning automaton (LA) [13].

LAs can be classified into two categories: *fixed-structure learning automata* (FSLA) and *variable structure learning automata* (VSLA). The LA is called fixed-structure if the following are fixed during the operation of the LA: (a) the probability of the transition from one state to another (the state transition function F) and (b) the action probability of any action in any state (the output function G). Examples of the fixed-structure LAs are $L_{2N,2}$, $G_{2N,2}$, Krylov, and Krinsky, which have been used in many applications such as adapting the back-propagation algorithm parameters [13].

Formally, a VSLA can be defined as six-tuple $(\Phi, \alpha, \beta, P, G, T)$, where P is the state probability distribution governing the choice of states at each instant t, T is the learning algorithm (also known as the learning scheme), and the remaining symbols correspond to the definition given above. For mathematical simplicity, it is assumed that each state corresponds to one distinct action. Thus $r = s$ and G is an identity mapping, i.e., G maps state Φ_i to action α_i for $i = 1 \ldots r$. Hence, we can speak interchangeably of states or actions and represent the VSLA as the quadruple (α, β, P, T), where:

(1) α is a set of actions.
(2) β is a set of inputs.
(3) $P = \left(p_1, p_2, \ldots, p_r\right)$ is the action probability distribution. The variable p_i is the probability of the action a_i being chosen and $\sum_{i=1}^{r} p_i = 1$.
(4) T is the learning algorithm that refers to a recurrence relation to update the action probability distribution. The action probability distribution is updated by the function $P(t + 1) = T[\alpha(t), \beta(t), P(t)]$ where $\alpha(t)$, $\beta(t)$ and $P(t)$ are the chosen action, the reinforcement signal (environment response), and the action probability distribution at instant t, respectively.

The key factor in a LA is choosing the learning algorithm. The simplest learning algorithm is the "*linear learning scheme*". Let $\alpha_i(t)$ be the chosen action by the LA and $\beta(t)$ be the environment response at instant t. The linear learning algorithm for P-model environments is given in Eqs. (1) and (2). If the chosen

action is rewarded by the environment ($\beta(t) = 0$), the action probability distribution is updated according to the Eq. (1), and if the chosen action is penalized by the environment ($\beta(t) = 1$), the action probability distribution is updated according to the Eq. (2).

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

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

In the above equations, a, b, and r are the reward parameter, the penalty parameter, and the number of actions, respectively. The parameters a and b determine the amount of increase and decrease in the action probabilities, respectively. If $a = b$, the Eqs. (1) and (2) are called *linear reward-penalty* (L_{RP}) algorithm, if $b = 0$, they are called *linear reward-inaction* (L_{RI}) algorithm, and if $0 < b << a < 1$, they are called *linear reward-ε penalty* ($L_{R\varepsilon P}$) algorithm.

From another viewpoint, LAs can also be classified into two categories: *finite action set learning automata* (FALA) and *continuous action set learning automata* (CALA) [14]. Equations (1) and (2) are used to update the action probability distribution in a FALA ($r < \infty$). In a CALA, the action probability distribution cannot be expressed by a finite vector of values. We should choose distributions that have a simple parametric form; the automaton then stores only these parameters. For example, for the Gaussian distribution, the automaton stores only the *mean* and *variance* of the distribution. The learning algorithm updates the mean and variance of the Gaussian distribution at any moment using Eqs. (3) and (4)

$$\mu_{t+1} = \mu_t - s\left(\frac{\Gamma_{x_t} - \Gamma_{\mu_t}}{g(\sigma_t)}\right)\left(\frac{x_t - \mu_t}{g(\sigma_t)}\right) \tag{3}$$

$$\sigma_{t+1} = \sigma_t - s\left(\frac{\Gamma_{x_t} - \Gamma_{\mu_t}}{g(\sigma_t)}\right)\left[\left(\frac{x_t - \mu_t}{g(\sigma_t)}\right)^2 - 1\right] + sC(\sigma_l - \sigma_t) \tag{4}$$

where

$$g(\sigma) = \begin{cases} \sigma_l, \sigma \leq \sigma_l \\ \sigma, \sigma > \sigma_l \end{cases} \tag{5}$$

and Γ is a noisy evaluation function, C is a sufficiently large positive constant, $s > 0$ is the step size for learning, and σ_l is a parameter of the algorithm, which is chosen to be sufficiently small to ensure that the obtained solution is close to the minimum of target function $f(\cdot)$, which is to be minimized.

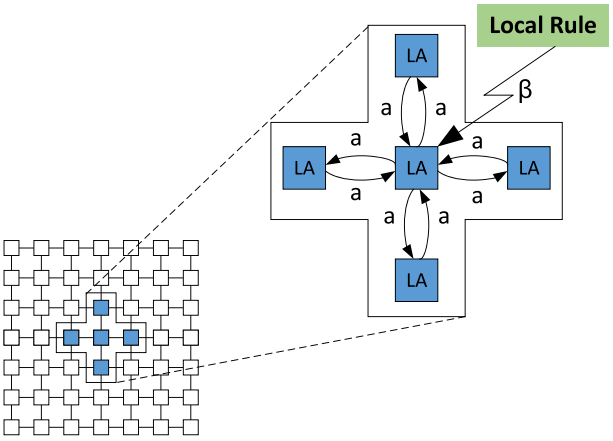


Fig. 3 Structure of the 2-dimensional cellular learning automaton

Cellular Learning Automata

Cellular learning automata (CLAs) [15] are mathematical models for dynamic complex systems that consist of large numbers of simple components. The simple components, which have learning capability, act together to produce complicated behavioral patterns. A CLA is a CA in which a LA is assigned to every cell. The LA residing in a particular cell determines the cell state based on its action probability distribution. Like CA, there is a rule that the CLA operates under. The rule of the CLA and the chosen actions by the neighboring LAs of any particular LA determine the reinforcement signal to the LA residing in a cell. The neighboring LAs of any particular LA constitute the local environment of that cell. The local environment of a cell is nonstationary because the action probability distributions of the neighboring LAs vary during the evolution of the CLA. This model is superior to CA because of its ability to learn and is superior to a single LA because it is a collection of LAs that can interact with each other. The basic idea of CLAs, a subclass of stochastic CAs, is to use LAs to adjust the state transition probability of stochastic CAs. In Fig. 3, the structure of the 2-dimensional CLA is shown.

The operation of a CLA can be described as follows: at the first step, the internal state of a cell is specified. The state of a cell is determined based on the action probability distribution of the LA residing in that cell. The action probability distribution of LAs is initialized based on past experience or at random. In the second step, the rule of the CLA determines the reinforcement signal to the LA residing in the cell. Finally, the LA updates its action probability distribution based on the received reinforcement signal and its chosen action. This process continues until the desired result is obtained. Formally, a d-dimensional CLA can be defined as quintuple $\left(Z^d, \Phi, A, N, F\right)$, where:

(1) Z^d is a lattice of d-tuples of integer numbers.
(2) Φ is a finite set of states.

(3) A is a set of LAs, each of which is assigned to one cell of the CLA.
(4) $N = \{n_1, n_2, \ldots, n_h\}$ is a finite subset of Z^d called the neighborhood vector, where $n_i \in Z^d$.
(5) $F = \Phi^{|N|} \rightarrow \beta$ is the local rule of the CLA, where β is the set of values that the reinforcement signal can take. It computes the reinforcement signal for each LA based on the chosen actions by the neighboring LAs.

CLAs can be classified into two categories: *synchronous cellular learning automata* (SCLA) and *asynchronous cellular learning automata* (ACLA). In a SCLA, all cells are synchronized with a global clock and executed at the same time, but in an ACLA, only some cells are activated[1] independently from each other. In some practical applications, the interaction between the CLA and the external environment is also considered. Such a CLA is known as an *open cellular learning automaton* (OCLA). In an OCLA, the evolution of the CLA depends not only on the local environment (neighboring cells) but also on the external environment. Because OCLA interacts with different types of environments, it can produce very complex patterns and behaviors.

Chaos Theory

Chaos theory is a branch of mathematics that studies dynamical systems that follow deterministic laws but appear random and unpredictable. The hallmark of chaotic systems is their sensitivity to initial conditions; a small change in the initial conditions of such systems results in a massive change in long-term behavior. However, chaotic systems are deterministic, not random. Given the same initial conditions, the long-term results are repeatable. A system must have the following properties to be considered as a chaotic system [16–18]:

Sensitivity to initial conditions This property indicates that a slight change in the initial conditions of a chaotic system produces quite different results in the long-term. Therefore, a long-term prediction of the behavior of chaotic systems cannot be made in detail. Long-term predictions can be made to some extent, as long as vast spatial scales are involved. Although a chaotic system is defined by a deterministic equation, the sensitivity to the initial conditions is the only reason for this unpredictability.

Topological mixing or topological transitivity (Ergodicity) This property can ensure chaotic variables traverse all states non-repeatedly within a certain range according to its own laws. Therefore, this property can be used as an optimization mechanism, ensuring that no solution is met twice and avoids being trapped in local optima.

Topological density (density of periodic orbits) This property expresses that every point in the search space is approached arbitrarily by periodic orbits.

[1] The process described in previous paragraph is performed only for active cells. In other words, only active cells update their status and action probability distribution.

In mathematics, chaotic maps are used to generate chaotic time series. One of the most commonly used chaotic maps is the *Logistic map*. The Logistic map is a second-order polynomial map with Eq. (6) [19]:

$$X_{n+1} = rX_n\left(1 - X_n\right) \tag{6}$$

obviously, $X \in [0, 1]$ holds under the condition that $X_0 \in (0, 1)$. In Eq. (6), r is a control parameter between 0 and 4 that determines the behavior of the variable X. The variable X can be *convergent*, *periodic*, or *chaotic*. When $0 \leq r \leq 3$, it converges to a constant value, when $3 < r \leq 3.56$, it shows periodic behavior, and when $3.56 < r \leq 4$, it shows chaotic behavior. In Fig. 4, the behavior of the variable X for different values of parameter r is shown.

A numerical approach to identify chaos can be developed based on the quantitative characterization of the divergence among the neighboring trajectories. If two trajectories that start close to each other deviate more and more as time increases, the system is said to be chaotic. The rate at which nearby trajectories deviate from each other with time is characterized by a quantity called the *Lyapunov exponent*. Negative Lyapunov exponents indicate convergence, while positive Lyapunov exponents demonstrate divergence and chaos. One can use the largest positive Lyapunov exponent (LLE) to detect the presence of chaos in a dynamical system. In the Logistic map, the largest positive Lyapunov exponent is obtained when r is 4 [20]. Therefore, in the proposed model, the parameter r of the Logistic map is set to 4.

Distributed Evolutionary Algorithms

EAs such as GA and PSO have been successfully applied to solve various optimization problems. The tradeoff between the exploration of new areas of the search space and the exploitation of good solutions is one of the key factors for the high performance of these algorithms compared to other metaheuristics. The balance between exploration and exploitation can be sharpened by some parameters of the EA, such as the population used (centralized or distributed), the variation operators applied, or the probability of applying them. In traditional EAs, there is a single population of individuals (centralized population) on which evolutionary operators are performed. In contrast, in distributed EAs, the population is somewhat decentralized. In most cases, the use of distributed populations provides a better sampling of the search space and thus improves both the numerical behavior and the execution time of the algorithm [21].

In general, distributed EAs are classified into two categories: *coarse-grained evolutionary algorithms* (island model) and *fine-grained evolutionary algorithms* (propagation model). In the coarse-grained EAs, the population is divided into several sub-populations, and each sub-population evolves independently of the other sub-populations using the traditional EA. Periodically migration operator is used to exchange a small number of individuals among the sub-populations. The migration rate, which controls how many individuals migrate, and the migration interval, which determines the frequency of migrations, are important issues to debate.

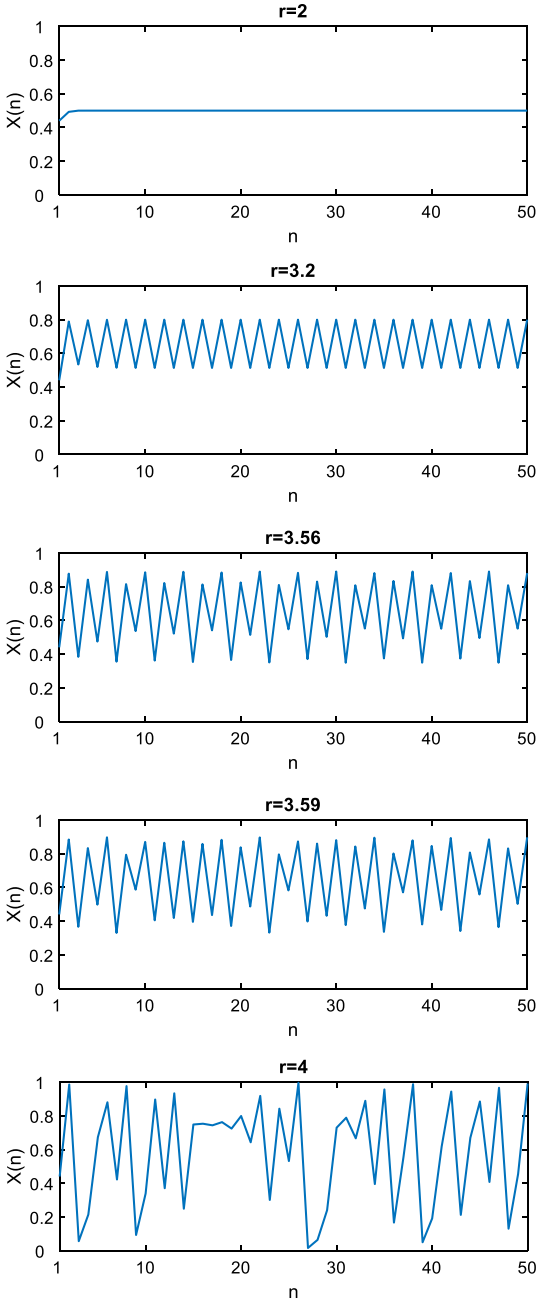


Fig. 4 The behavior of the variable X for different values of parameter r in the Logistic map

Each sub-population is often referred to as an *island*. On the other hand, in the fine-grained EAs, each individual of the population is assigned to a cell of a cellular lattice. Therefore, fine-grained EAs are also called cellular EAs. The population is

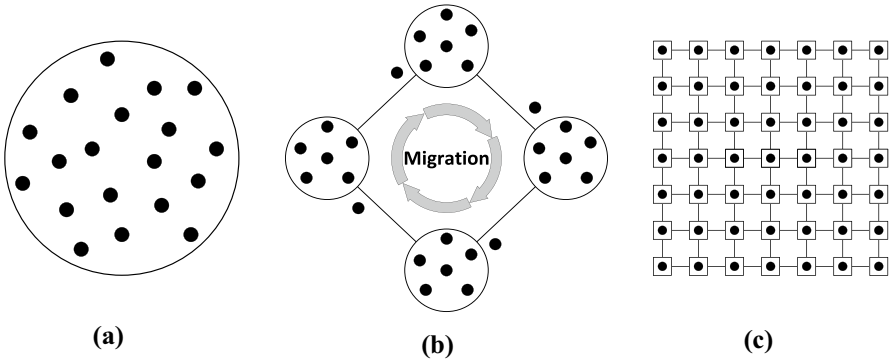


Fig. 5 Population structure in evolutionary algorithms. **a** Traditional evolutionary algorithm, **b** coarse-grained evolutionary algorithm, and **c** fine-grained evolutionary algorithm. Each small black circle represents an individual of the population

regarded as a system of active individuals in which each individual interacts only with their neighbors. That is, when producing a new individual at a cell, parents are selected from its neighboring individuals, including itself. Fitness evaluation is performed simultaneously for all the individuals, and the selection of individuals for reproduction and mating takes place locally within the neighborhood. In this manner, it can be expected that the cellular EAs can maintain more diversity in the population than the traditional EAs. Information is slowly diffused in the cellular lattice, forming clusters of solutions around different optimal solutions. Using these two models, it is possible to implement EAs in parallel. In the island model, each sub-population, and in the propagation model, each individual can be run on a processing element independent of the other processing elements. Each processing element only interacts locally with other processing elements. In Fig. 5, the population structure of the different EAs is shown [21–24].

Evolutionary Model Based on Cellular Learning Automata and Chaos Theory

EAs are one of the common methods for solving NP-hard optimization problems. These algorithms have two major problems: (a) they may be trapped in local optima, and (b) they have difficulties in addressing large-scale problems effectively. To overcome the problem (a), various new EAs such as combining mutation operators [25], macroevolutionary algorithm [26], immune genetic algorithm [27], orthogonal genetic algorithm [28], and microgenetic algorithm [29] have been proposed. These algorithms proved to be effective and boosted the development of EAs. To overcome problem (b), distributed EAs (*coarse-grained* and *fine-grained* EAs) [21–24] have been proposed. Cellular EAs are a kind of technique that allows the parallel implementation of the fine-grained EAs. Cellular EAs also suffer the same problem of premature convergence of traditional EAs.

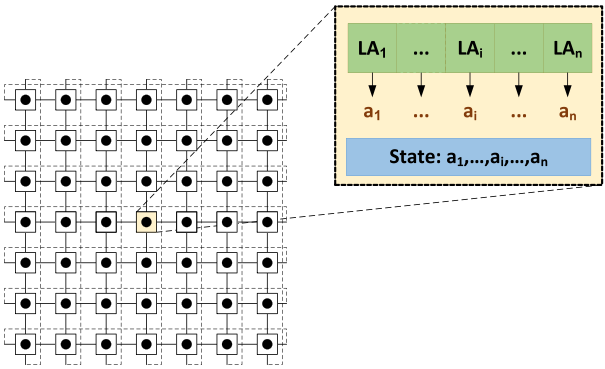


Fig. 6 Structure of a cell in the CCLA-EM model

In this section, a new evolutionary model is proposed to overcome both problems (a) and (b). In the proposed model, a combination of an EA with a CLA is used. For the sake of simplicity, suppose that we have a n-variable optimization problem $f(x_1, \ldots, x_n), x_i^l \le x_i \le x_i^u$. The goal is to find values for variables x_1, …, x_n so that the function f is optimal for those values. In the proposed model, entitled evolutionary model based on cellular learning automata and chaos theory (CCLA-EM), every individual (solution) of the population consists of n elements in which the value of the ith element indicates the value of the variable x_i of the function f. In the CCLA-EM, each individual of the population is mapped to a cell of a 2-dimensional CLA. Moreover, n LAs are assigned to each cell. The variable n is the number of optimization problem variables. The chosen actions by the LAs residing in a particular cell determine the state (value of the optimization problem variables) of that cell. In Fig. 6, the structure of a cell in the CCLA-EM is shown. As you can see, n learning automata LA_1, …, LA_n are assigned to the cell. Suppose that the chosen actions by the n learning automata at step t are a_1, …, a_n, respectively. The state of the cell at step t is equal to "a_1, …, a_n" which is specified by concatenating the chosen actions by all the LAs residing in that cell. The chosen action by the learning automaton LA_i or the value of ith element of the cell state (i.e., a_i) determines the value of the variable x_i of the optimization problem. In other words, the cell state indicates a solution to the optimization problem. So, in what follows, the words "cell state" and "solution" are used interchangeably. Suppose that L_{row} and L_{col} indicate the number of rows and columns of the cellular lattice, respectively. These two parameters are the inputs of CCLA-EM. In what follows, the steps of the CCLA-EM are explained (see pseudo-code of Fig. 7).

Step 1) Initialization First, a torus cellular lattice is created with L_{row} rows and L_{col} columns like Fig. 6. Then, n LAs are assigned to each cell, and their action probability distribution is initialized. Supposing that the number of actions of a LA is r; its action probability distribution is initialized as $P = \left\{ p_i = \frac{1}{r} \right\}, i = 1, \ldots, r$. Finally, the state of each cell is determined based on the chosen actions of the LAs residing in that cell. Since the state of a cell

Algorithm CCLA-EM /***C**haotic **C**ellular **L**earning **A**utomata based **E**volutionary **M**odel*/

1. **Inputs:**
2. optimization problem $f(x_1, \ldots, x_n), x_i^l \le x_i \le x_i^u$ (n denotes the number of variables)
3. Lrow, Lcol: lattice size
4. *N: neighborhood vector*
5. maxIter: maximum number of iterations
6. maxStallIter: maximum number of stall iterations
7.
8. **Output:**
9. bestSol: best solution found by the CCLA-EM
10.
11. **begin**
12. CreateCellularLattice&AssignLAsToCells(Lrow, Lcol, f);
13. InitializeLAsActionProbabilityDistribution&CellStates(f);
14. CalculateFitnessOfEachCellSolution();
15. iter←0;
16. stallIter←0;
17. **while !** IsTerminate(iter, stallIter, maxIter, maxStallIter)
18. iter←iter+1;
19. **foreach** cell C_i in the cellular lattice /*this can be done synchronously or asynchronously*/
20. /*CLA phase*/
21. neighbors←*GetNeighbors*(C_i, N);
22. actions←SelectActions(C_i); /*each LA in cell C_i select an action*/
23. βs←CalculateReinforcementSignalForEachLAInCellCi(actions, neighbors);
24. UpdateActionProbabilityDistributionOfEachLAInCellCi(βs);
25. **if** Fitness(new solution of cell C_i) > Fitness(old solution of cell C_i)
26. C_i.state← new solution of cell C_i; /*old solution is replaced with new one*/
27. C_i.fitness←Fitness(new solution of cell C_i);
28. **end**
29.
30. /*evolutionary phase*/
31. parents←SelectParents(neighbors);
32. newSol←ApplyEvolutionaryOrVariationOperatorsOfAnEvolutionaryAlgorithm(parents);
33. **if** Fitness(newSol) > Fitness(solution of cell C_i)
34. C_i.state← newSol;
35. C_i.fitness←Fitness(newSol);
36. ReinitializeActionProbabilityDistributionOfEachLAInCellCi(parents);
37. **end**
38. **end** of foreach
39.
40. UpdateBestSolution&StallIter();
41. **end** of while
42.
43. **return** bestSol;
44. **end** of algorithm

Fig. 7 Pseudo-code of the CCLA-EM evolutionary model

indicates an individual of the population, population size is equal to the number of cells, i.e., $L_{\text{row}} \times L_{\text{col}}$ (the rows 12 and 13 in pseudo-code of Fig. 7).

Step 2) Evaluating the solutions Using an objective function, a fitness value is assigned to the solution of each cell. The fitness values are used in other steps of the CCLA-EM (row 14 in the pseudo-code of Fig. 7).

Step 3) Checking the terminating condition The following phases (cellular learning automaton phase and evolutionary phase) are repeated for all the cells synchronously or asynchronously until a terminating condition is satisfied. The CCLA-EM is terminated when it reaches the maximum number of predetermined generations, or the fitness of the best solution has not been changed in several successive generations. In the following, the cellular learning automaton and evolutionary phases are explained for a given cell, C_i (row 17 in the pseudo-code of Fig. 7).

Step 4) Cellular learning automaton phase This phase includes two steps: (4–1) choosing an action and updating the action probability distribution and (4–2) updating the solution.

Step 4–1) Choosing an action and updating the action probability distribution Every LA residing in the cell C_i chooses an action based on its action probability distribution. The chosen action by each LA is evaluated by the local environment (LAs residing in the neighboring cells), and a reinforcement signal is generated for it. Each LA updates its action probability distribution based on the received reinforcement signal and its chosen action. Evaluating the chosen action of a LA by the local environment depends on the optimization problem at hand. For example, majority voting can be used to evaluate the chosen action by an automaton. In the majority voting, if the chosen action by the majority of the neighbors is the same as the chosen action by an automaton, the chosen action will be rewarded and otherwise penalized (the rows 21–24 in pseudo-code of Fig. 7).

Step 4–2) Updating the solution A new solution is obtained by concatenating the chosen actions of the LAs residing in the cell C_i. If the fitness of the new solution is more than the fitness of the previous solution, it replaces the previous solution (the rows 25–28 in the pseudo-code of Fig. 7).

Step 5) Evolutionary phase This phase includes three steps: (5–1) selecting parents, (5–2) applying evolutionary or variation operators, and (5–3) replacement.

Step 5–1) Selecting parents Several cells are selected among the neighboring cells of the cell C_i as parents. Different types of the neighborhood are shown in Fig. 1. The chance for a cell to be chosen as a parent is proportional to its solution fitness. In other words, the cell whose solution fitness is higher has a better chance of being selected. To select parent cells, we can use one of the existing methods like roulette wheel selection. The number of cells which must be selected as the parents depends on the EA applied in the evolutionary phase. For instance, in the GA, two and in the PSO, one parent is selected (row 31 in the pseudo-code of Fig. 7).

Step 5–2) Applying evolutionary or variation operators According to the EA applied in the evolutionary phase, the evolutionary or variation operators are done on the parent(s). By doing this, a new solution is generated. For example, if the EA applied in the evolutionary phase is the GA, a new solution is generated by applying recombination and mutation operators on the selected parents, and if the EA applied in the evolutionary phase is the PSO, a new solution is generated by applying updating equation of the PSO on the selected parent (row 32 in the pseudo-code of Fig. 7).

Step 5–3) Replacement The fitness of the generated solution in the previous step is compared with the fitness of the cell C_i solution. If its fitness is higher than the fitness of the cell C_i solution, it replaces the cell C_i solution and the action probability

distribution of every LA residing in the cell C_i is reinitialized according to the action probability distribution of the corresponding LAs in the parent cells (the rows 33–37 in pseudo-code of Fig. 7).

If all cells are activated simultaneously, CCLA-EM is called synchronous, and if cells are activated in a particular order, CCLA-EM is called asynchronous. Obviously, the behavior of the CCLA-EM depends on how the cells are activated and the size and shape of the neighborhood. Moreover, the visiting order of the cells in the asynchronous case is also an important issue in the behavior of the CCLA-EM. In the asynchronous CCLA-EM, the cells can be activated in one of the following ways:

- *Line Sweep (LS)* This is the simplest method. It lies in sequentially updating the cells row by row and from left to right.
- *Fixed Random Sweep (FRS)* In this case, the next cell to activate is selected with a uniform probability without replacement. That is, it is not possible to activate one cell twice in an iteration. A fixed permutation is used for all the iterations.
- *New Random Sweep (NRS)* This is similar to FRS, with the difference of using a new random permutation of cells in each iteration.
- *Uniform Choice (UC)* In this method, the next cell to activate is randomly selected with uniform probability among all the lattice cells with replacement. So, it is possible to activate one cell more than once in the same iteration.

It has been shown that using chaotic numbers instead of random ones leads to a significant improvement in the convergence and performance of an EA [9–11]. Generally, it can be said that using the chaotic numbers instead of random ones in the EAs will avoid trapping in local optima and, consequently, premature convergence. So, we can also use the chaotic numbers instead of random ones in different steps (steps 1, 4–1, 5–1, 5–2, and also in the asynchronous selection of the cells) of the CCLA-EM. We can use one of the chaotic maps, like the Logistic map, to generate the chaotic time series.

Application of the CCLA-EM on the Community Structure Detection Problem

In this section, the application of the CCLA-EM on the community structure detection problem in complex networks is investigated, and its performance is evaluated and compared with that of GA-net [30] and ICLA-net [31]. A complex network can be represented by a graph $G = (V, E)$, in which V is a set of n nodes, and $E \subseteq V \times V$ is a set of edges. For example, in the graph of a social network, which is a kind of complex network, the nodes represent individuals, and the edges represent the relationship between individuals [32]. Community structure is one of the most important topological characteristics of complex networks. Detecting community structure is a highly challenging problem in analyzing complex networks, and it has high significance for understanding the function and organization of complex networks. The purpose of the community structure detection is to partition set V (nodes of a

complex network) into k disjoint sets (each set is called a group or community) C_1, …, C_k so that:

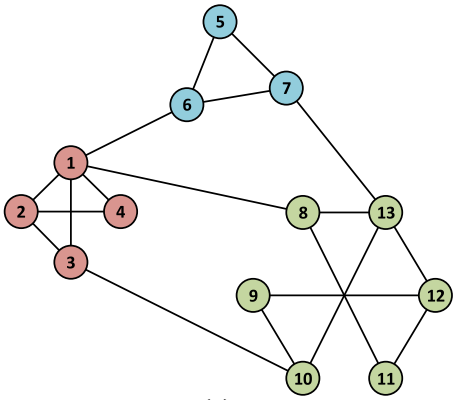
a) $\bigcup_{i=1}^{k} C_i = V$,
b) The density of the edges is more within the communities and less between the communities. Modularity [33] is a criterion that compares the density of edges within and between communities with the null model. The null model is a random graph with the same size and degree distribution as the original graph (a graph that we want to find its community structure). The higher the modularity value, the higher the density of edge within the communities and lower between the communities than the null model. In other words, the community structure is of high quality. The mathematical representation of the modularity is given in Eq. (7).

In general, a community is defined as a group of nodes with dense intra-group interactions and relatively sparse inter-group interactions. Since communities of a complex network correspond to the operational units of the underlying system, detecting communities can give valuable information about the function and organization of complex systems [34–36]. Modularity is a criterion for comparing the quality of the obtained communities through different algorithms. It is also used as an objective function for optimization. According to the modularity, the community structure detection problem can be modeled as a modularity optimization problem. It has been proven that modularity optimization is an NP-hard problem [37–39].

In the community structure detection problem, optimization function $f(x_1, \ldots, x_n)$ is the modularity function that must be maximized, and variables $x_1, \ldots, x_n$ are the community identifier of nodes $1, \ldots, n$ of the input graph with n nodes, respectively.

Solution Representation

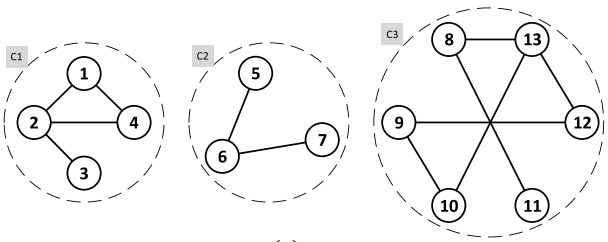
In this paper, locus-based adjacency representation [40] is used for the community structure detection problem. In this representation, a solution of the population consists of n elements g_1, …, g_n. The variable n is the number of nodes in a complex network given as input. Element g_i of the solution vector can take ID (identifier) of one of the nodes adjacent to node i (including node i itself). In this representation, each solution is a graph with the same nodes as the input complex network, and an edge exists between two nodes i and j if ith element of the solution has the value j. The decoding process of this representation is to identify all the connected components of the graph. The nodes belonging to the same connected component establish a community. Using a breadth-first search, the decoding process can be done in linear time $O(|V| + |E|)$. The main advantage of this representation is that there is no need to fix the number of communities in advance, as it is automatically determined in the decoding process. For example, consider the toy network of Fig. 8a. It consists of thirteen nodes numbered from 1 to 13 and three communities. Various communities are colored by different colors. A sample solution vector for this network is given in Fig. 8b. Each element g_i of the solution vector has a value equal to the ID



(a)

Node	1	2	3	4	5	6	7	8	9	10	11	12	13
Solution Vector	2	4	2	1	6	5	6	13	12	9	8	13	10

(b)



(c)

Fig. 8 **a** A toy network with thirteen nodes and three communities, **b** a sample solution vector, and **c** the community structure corresponding to the solution vector

of one of the nodes adjacent to node i. For instance, the 6th element of the solution vector has a value of 5 (one of the neighbors of node 6). The community structure corresponding to the solution vector is shown in Fig. 8c.

Modeling of the Community Structure Detection Problem to the CCLA-EM

To solve the community structure detection problem using the CCLA-EM algorithm, n learning automata LA_1, …, LA_n are assigned to each cell. The set of doable actions by the learning automaton LA_i is equal to the set of neighboring nodes of node i (including node i itself). The chosen action by the learning automaton LA_i determines the value of the ith element of the solution vector. The chosen actions by all the LAs residing in a particular cell determine the state (solution vector) of that cell. For example, for the graph of Fig. 8a, $n = 13$ LAs are assigned to every

Table 1 The value used for the parameters of algorithms CCLA-EM, GA-net, and ICLA-net

CCLA-EM parameter	Value
Lattice size	5×5
Neighborhood type	von-Neumann
Cells updating mode	synchronous
LAs reward parameter	0.1
Recombination rate	0.8
Recombination operator	One-way crossover
Mutation rate	0.2
Mutation operator	Gene-based mutation[a]
Maximum number of stall generations	100
Maximum number of generations	500
GA-net parameter	**Value**
Population size	100
Crossover rate	0.8
Mutation rate	0.2
Exponent α	1.5
Maximum number of generations	100
ICLA-net parameter	**Value**
LAs reward parameter	0.1
Number of initial iterations	20
Maximum number of iterations	500

[a]"Gene-based mutation" is a generalization of "bitwise mutation" for the community structure detection problem. In the bitwise mutation, one or more genes are randomly selected and their value changed. In the gene-based mutation used for the community structure detection problem, a gene is randomly selected and its value changed according to the locus-based adjacency representation

cell and the set of doable actions by the learning automaton LA_6 is equal to the set $\{6, 1, 5, 7\}$.

Parameter Setting

In all experiments of this section, GA is used in the evolutionary phase of the CCLA-EM algorithm. In Table 1, the value used for the parameters of algorithms CCLA-EM, GA-net, and ICLA-net is given.

Evaluation Criteria

Many criteria have been presented to evaluate the quality of the obtained community structure by an algorithm. The most well-known criteria are *modularity* (Q) [33] and *normalized mutual information* (NMI) [41]. Modularity is a criterion for comparing

the quality of the obtained communities through different algorithms. Modularity criterion is defined as follows:

$$Q = \frac{1}{2m} \sum_{v,w \in V} \left[A_{vw} - \frac{k_v k_w}{2m} \right] \delta(c_v, c_w) \tag{7}$$

where A is the adjacency matrix of the network; $m = \frac{1}{2}\sum_{vw} A_{vw}$ is the total number of network edges; k_v is the degree of node v; c_v is the community of node v and $\delta(i,j) = 1$ if $i = j$, otherwise $\delta(i,j) = 0$. The value of Q lies in the interval $[-0.5, 1]$. Larger values of Q indicate more connections inside the communities than the expected number in the *null model*. A null model is a network that matches the original network in certain structural features but whose other features are essentially randomly determined. In the problem of community structure detection, the null model is a random network (the edges are placed randomly between the nodes) with the same number of nodes, the same number of edges, and the same degree distribution as the original network.

Normalized mutual information criterion operates according to a matrix called *confusion matrix*. The rows of the confusion matrix correspond to the ground-truth communities, and its columns correspond to the found communities. Element n_{ij} of the confusion matrix indicates the number of common nodes in the ith ground-truth community and jth found community. Let A and B be two different partitions of a network with n nodes, $c_A(c_B)$ indicates the number of communities in partition $A(B)$ and $n_{i.}(n_{.j})$ denotes the sum of the elements of the ith row (jth column) of the confusion matrix. Based on information theory, *NMI* criterion is defined as follows:

$$\mathrm{NMI}(A,B) = \frac{-2\sum_{i=1}^{c_A}\sum_{j=1}^{c_B} n_{ij} \log\left(\frac{n_{ij} n}{n_{i.} n_{.j}}\right)}{\sum_{i=1}^{c_A} n_{i.} \log\left(\frac{n_{i.}}{n}\right) + \sum_{j=1}^{c_B} n_{.j} \log\left(\frac{n_{.j}}{n}\right)} \tag{8}$$

The value of *NMI* lies in the interval [0, 1]. Larger values of NMI indicate that the found communities are more matched with the ground-truth communities.

Benchmark Networks

Real-world and synthetic benchmark networks are used to evaluate the performance of the CCLA-EM and compare it with other algorithms. The LFR generator [42] is used to generate synthetic benchmark networks. In LFR benchmark networks, both node degrees and community sizes follow the power-law distribution with exponents τ_1 and τ_2, respectively. In these networks, a parameter namely mixing parameter, μ, is used to control the community structure in the network. The mixing parameter μ determines the average of each node's connections to the other communities. When μ is small, the community structure in the network is significant and clear. It gradually becomes unclear with the increase of μ. In Tables 2 and 3, general information of the used real-world and synthetic benchmark networks is given, respectively.

Table 2 General information of the real-world benchmark networks

Network	Description	Node	Edge
Karate	Zachary's karate club	34	78
Contiguous	The USA states contiguous network	49	107
Dolphins	Dolphins social network	62	159
PolBooks	Books about US politics	105	441
AdjNoun	Copperfield word adjacencies network	112	425
Football	American college football network	115	613
Jazz	Jazz musicians' network	198	2742
Email	URV email network	1133	5451
Facebook	EGO-Facebook friendship network	2888	2981

Table 3 General information of the LFR benchmark networks

Parameter	Description	Value
N	Number of nodes	500
k	Average degree	25
max$k$	Maximum degree	50
μ	Mixing parameter	{0.0:0.05:0.5}
τ_1	Minus exponent for the degree distribution	2
τ_2	Minus exponent for the community size distribution	1
min$c$	Minimum for the community sizes	50
max$c$	Maximum for the community sizes	100

Effect of Cellular Lattice Size on the Performance of the CCLA-EM Algorithm

In this section, the effect of cellular lattice size on the performance of the CCLA-EM algorithm is investigated. In Fig. 9, the average results obtained from the CCLA-EM algorithm on the LFR benchmark networks for different values of the cellular lattice size are shown. Each data is an average of 10 runs. As you can see, by increasing the size of the cellular lattice (the number of population individuals), the performance of the CCLA-EM algorithm increases for both Q and NMI criteria. This increase is significant up to 5×5 size, but beyond that, the performance of the CCLA-EM algorithm does not improve significantly. Since the execution time of the CCLA-EM algorithm increases by raising the size of the cellular lattice, there must be a compromise between the performance and execution time. According to the results of Fig. 9, we can say that the ideal size for the cellular lattice is 5×5. In other words, when the size of the cellular lattice is 5×5, the CCLA-EM algorithm achieves a desirable solution in a reasonable time. The size of the cellular lattice is considered as 5×5 in all the results reported in the rest of the paper. It is worth noting that the execution time of the CCLA-EM algorithm increases as linear by increasing the size of the cellular lattice.

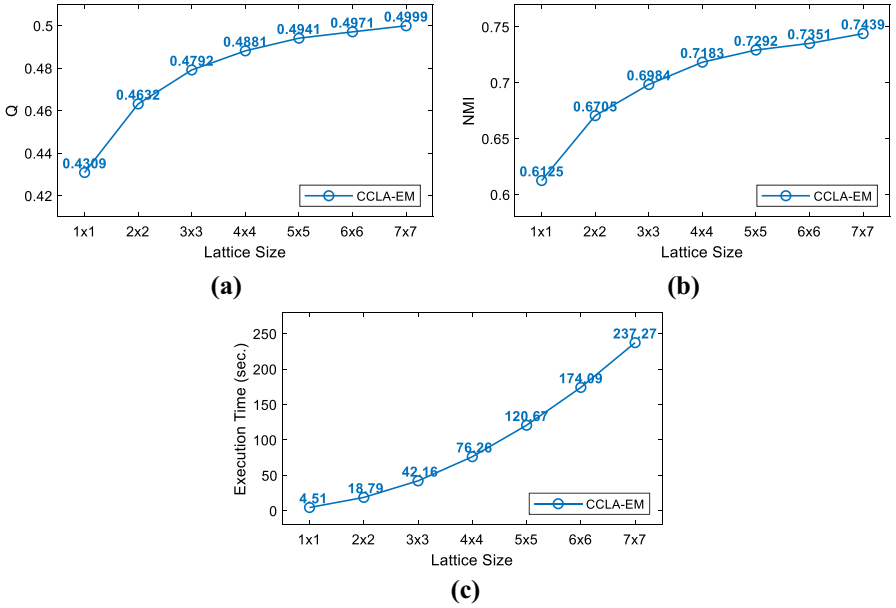


Fig. 9 Effect of cellular lattice size on the performance of the CCLA-EM algorithm. **a** Effect of cellular lattice size on the criterion Q, **b** effect of cellular lattice size on the criterion NMI, and **c** effect of cellular lattice size on the execution time

For example, the required time for converging the CCLA-EM algorithm with a 6×6 cellular lattice (36 cells) is almost four times more than the required time for converging the CCLA-EM algorithm with a 3×3 cellular lattice (9 cells) (see Fig. 9c). So, by parallel implementation and running of the CCLA-EM algorithm on several processing elements, its execution time is decreased as Eq. (9). CO is the communication overhead or the required time for communication between the processing elements, which can be ignored.

$$\text{Parallel Execution Time} = \frac{\text{Serial Execution Time}}{\text{Number of Processing Elements}} + \text{CO} \tag{9}$$

Effect of Using Chaotic Numbers on the Performance of the CCLA-EM Algorithm

In this section, the effect of using chaotic numbers on the performance of the CCLA-EM algorithm is investigated. In Fig. 10, the effect of using chaotic numbers instead of random ones on the performance of the CCLA-EM algorithm is shown. Each data is the average of 10 runs of the CCLA-EM algorithm with 5×5 cellular lattice on the LFR benchmark networks. As you can see, the proposed algorithm has better performance in both Q and NMI criteria using the chaotic numbers (CCLA-EM) than the proposed algorithm in which the random numbers are used (CLA-EM). The

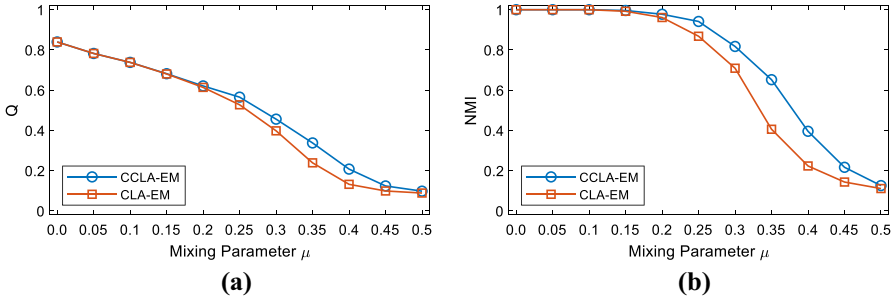


Fig. 10 Effect of using chaotic numbers on the performance of the CCLA-EM algorithm. **a** Effect of using chaotic numbers on the criterion Q and **b** effect of using chaotic numbers on the criterion NMI

performance of both algorithms is almost the same for smaller values of the mixing parameter ($\mu \leq 0.2$). However, by increasing μ, the CCLA-EM algorithm performs much better than the CLA-EM algorithm by optimal searching of the search space.

Comparing the Performance of the CCLA-EM Algorithm with Other Algorithms

In this section, the performance of the CCLA-EM algorithm on the community structure detection problem is compared with that of GA-net [30] and ICLA-net [31]. The reason for choosing these two algorithms for comparison is that the ICLA-net algorithm uses the CLA similar to the first phase of the CCLA-EM algorithm, and the GA-net algorithm uses the GA similar to the second phase of the CCLA-EM algorithm for solving the community structure detection problem. In the GA-net algorithm, the criteria *community score* is introduced to evaluate the quality of a community structure, and the genetic algorithm is used to optimize it. In this algorithm, the locus-based adjacency representation is used. Moreover, in this algorithm, specialized variation operators (crossover and mutation) are used. These operators reduce the search space by generating safe solutions. Therefore, the convergence of the algorithm is improved. In a safe solution, the ith gene can have a value j if and only if there is an edge between nodes i and j. In the ICLA-net algorithm, the entire complex network is modeled using an irregular CLA, and the community structure is found through the evolution of the CLA. In this algorithm, each node of the network is mapped to a cell of the irregular CLA, and a LA is assigned to every cell. The chosen action by the LA residing in a particular cell determines the state of that cell. The state of a cell represents the value of its corresponding node in the locus-based adjacency representation.

In Fig. 11, the average Q, NMI, and execution time obtained from different algorithms on the LFR benchmark networks are shown. Each data is an average of 10 runs. As you can see, the CCLA-EM algorithm outperforms both GA-net and ICLA-net algorithms for all the values of μ in terms of both criteria Q and NMI. This is because the CCLA-EM algorithm is not trapped in a local optima, and it can find high-quality solutions by overcoming the premature convergence problem by combining the CLA and the GA as well as using the chaotic numbers. Also, by

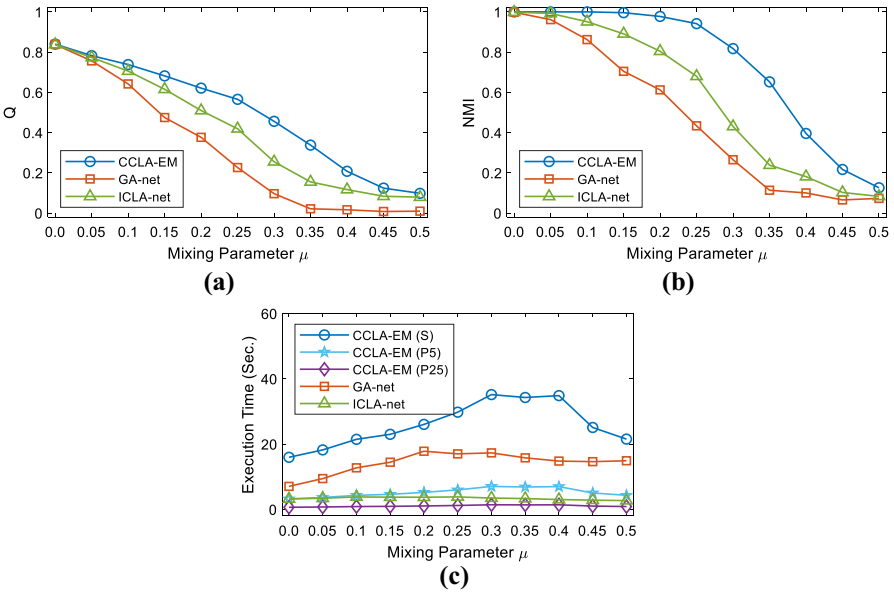


Fig. 11 The average Q, NMI, and execution time obtained from different algorithms on the LFR benchmark networks

increasing μ, the performance of all three algorithms decreases due to increasing ambiguity in the community structure. As you can see, this decrease in the CCLA-EM algorithm is much smaller than algorithms GA-net and ICLA-net. In Fig. 11c, the execution time of the CCLA-EM algorithm in different scenarios is compared with that of GA-net and ICLA-net. For the CCLA-EM algorithm, three scenarios CCLA-EM(S), CCLA-EM(P5), and CCLA-EM(P25) are considered. In CCLA-EM(S), the number of processing elements is considered 1. In other words, the serial execution time of the algorithm is given. In CCLA-EM(P5) and CCLA-EM(P25), the number of processing elements is considered 5 and 25, respectively. In other words, the parallel execution time of the algorithm with 5 and 25 processing elements is given, respectively. As you can see, in the case of CCLA-EM(S), the execution time of the CCLA-EM algorithm is longer than both algorithms GA-net and ICLA-net. In the case of CCLA-EM(P5), the execution time of the CCLA-EM algorithm is less than the algorithm GA-net and longer than the algorithm ICLA-net. In the case of CCLA-EM(P25), the execution time of the CCLA-EM algorithm is less than both algorithms GA-net and ICLA-net. In other words, it can be said that with parallel implementation and using enough processing elements, the execution time of the CCLA-EM algorithm will be less than both algorithms GA-net and ICLA-net.

In Fig. 12, the average Q obtained from different algorithms on the real-world benchmark networks is shown. Each data is an average of 10 runs. As you can see, the CCLA-EM algorithm outperforms both GA-net and ICLA-net algorithms. Since the ground-truth community structure of most real-world benchmark networks is unknown, the NMI criterion is not given for these networks. In Fig. 13, an instance

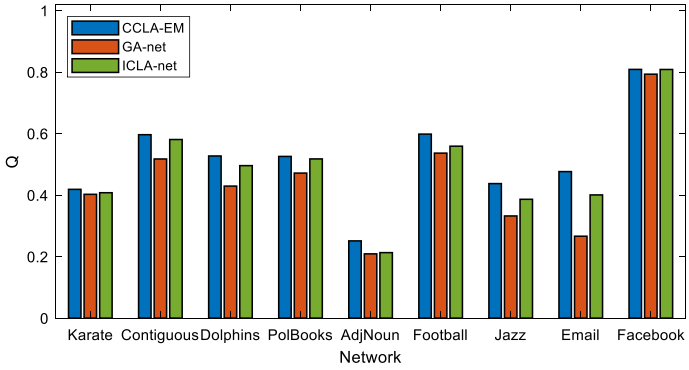


Fig. 12 The average Q obtained from different algorithms on the real-world benchmark networks

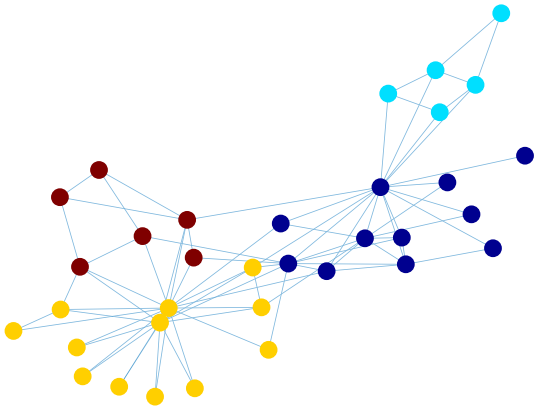


Fig. 13 An instance of the obtained community structure by the CCLA-EM algorithm on the Karate network

of the obtained community structure by the CCLA-EM algorithm on the Karate network is shown.

Conclusion and Future Works

In this paper, a new evolutionary model, called CCLA-EM, is proposed to solve optimization problems. The proposed model greatly overcomes the premature convergence problem of the existing EAs by combining an EA with a CLA and using chaotic numbers instead of random ones. Experiments on various benchmarks of the community structure detection problem indicate the superiority of the proposed model to the well-known algorithms GA-net and ICLA-net. In this paper, the linear learning algorithm is used in the CLA phase, and the GA is applied in the evolutionary phase of the proposed model for solving the community structure detection problem in complex networks. Using other learning algorithms like Q-learning in the CLA phase and other EAs like PSO in the

evolutionary phase of the proposed model and evaluating the performance of the proposed model in other applications and problems will be the focus of our work in the future.

Data Availability The data that support the findings of this study are available from the corresponding author upon reasonable request.

References

1. Zelinka, I., Snasael, V., Abraham, A.: Handbook of optimization: from classical to modern approach. Springer (2012)
2. Tenne, Y., Goh, C.-K.: Computational intelligence in expensive optimization problems. Springer (2010)
3. Alba, E., Blum, C., Asasi, P., Leon, C., Gomez, J.A.: Optimization techniques for solving complex problems. Wiley (2009)
4. Korte, B., Vygen, J., Korte, B., Vygen, J.: Combinatorial optimization. Springer (2012)
5. Blum, C., Roli, A.: Metaheuristics in combinatorial optimization: overview and conceptual comparison. ACM Comput. Surv. **35**(3), 268–308 (2003)
6. Liu, J., Zhong, W., Jiao, L.: A multiagent evolutionary algorithm for combinatorial optimization problems. IEEE Trans. Syst. Man Cybern. Part B (Cybern.) **40**(1), 229–240 (2009)
7. Liu, J., Zhong, W., Jiao, L.: A multiagent evolutionary algorithm for constraint satisfaction problems. IEEE Trans. Syst. Man Cybern. Part B (Cybern.) **36**(1), 54–73 (2006)
8. Zhong, W., Liu, J., Xue, M., Jiao, L.: A multiagent genetic algorithm for global numerical optimization. IEEE Trans. Syst. Man Cybern. Part B (Cybern.) **34**(2), 1128–1141 (2004)
9. Juan, L., Zixing, C., Jianqin, L.: Premature convergence in genetic algorithm: analysis and prevention based on chaos operator. In: Proceedings of the 3rd World congress on intelligent control and automation (Cat. No. 00EX393), vol. 1. IEEE, pp. 495–499 (2000)
10. Caponetto, R., Fortuna, L., Fazzino, S., Xibilia, M.G.: Chaotic sequences to improve the performance of evolutionary algorithms. IEEE Trans. Evol. Comput. **7**(3), 289–304 (2003)
11. Zarei, B., Meybodi, M.R., Masoumi, B.: Chaotic memetic algorithm and its application for detecting community structure in complex networks. Chaos **30**(1), 013125 (2020)
12. Wolfram, S.: Statistical mechanics of cellular automata. Rev. Mod. Phys. **55**(3), 601 (1983)
13. Narendra, K.S., Thathachar, M.A.: Learning automata: an introduction. Courier corporation (2012)
14. Santharam, G., Sastry, P., Thathachar, M.: Continuous action set learning automata for stochastic optimization. J. Franklin Inst. **331**(5), 607–628 (1994)
15. Beigy, H., Meybodi, M.R.: A mathematical framework for cellular learning automata. Adv. Complex Syst. **7**(03n04), 295–319 (2004)
16. Lorenzelli, F.: The essence of chaos. CRC Press (2014)
17. Smith, P.: Explaining chaos. Cambridge University Press (1998)
18. Williams, G.: Chaos theory tamed. CRC Press (1997)
19. Ausloos, M., Dirickx, M.: The logistic map and the route to chaos: From the beginnings to modern applications. Springer (2006)
20. Hilborn, R.C.: Chaos and nonlinear dynamics: an introduction for scientists and engineers. Oxford University Press (2000)
21. Alba, E., Dorronsoro, B.: Cellular genetic algorithms. Springer (2009)
22. Li, X., Wu, J., Li, X.: Theory of practical cellular automaton. Springer (2018)
23. Sudholt, D.: Parallel evolutionary algorithms. In: Springer handbook of computational intelligence, pp. 929–959. Springer (2015)
24. Tomassini, M.: Parallel and distributed evolutionary algorithms: A review. In: Miettinen, K., Makela, M., Neittaanmaki, P., Periaux, J. (eds.) Evolutionary Algorithms in Engineering and Computer Science, pp. 113–131. John Wiley & Sons, LTD, New York (1999)

25. Chellapilla, K.: Combining mutation operators in evolutionary programming. IEEE Trans. Evol. Comput. **2**(3), 91–96 (1998)
26. Marin, J., Sole, R.V.: Macroevolutionary algorithms: a new optimization method on fitness landscapes. IEEE Trans. Evol. Comput. **3**(4), 272–286 (1999)
27. Jiao, L., Wang, L.: A novel genetic algorithm based on immunity. IEEE Trans. Syst. Man Cybern. Part A Syst. Hum. **30**(5), 552–561 (2000)
28. Leung, Y.-W., Wang, Y.: An orthogonal genetic algorithm with quantization for global numerical optimization. IEEE Trans. Evol. Comput. **5**(1), 41–53 (2001)
29. Kazarlis, S.A., Papadakis, S.E., Theocharis, J., Petridis, V.: Microgenetic algorithms as generalized hill-climbing operators for GA optimization. IEEE Trans. Evol. Comput. **5**(3), 204–217 (2001)
30. Pizzuti, C.: GA-net: a genetic algorithm for community detection in social networks. In: International conference on parallel problem solving from nature, Springer, pp. 1081–1090 (2008)
31. Zhao, Y., Jiang, W., Li, S., Ma, Y., Su, G., Lin, X.: A cellular learning automata based algorithm for detecting community structure in complex networks. Neurocomputing **151**, 1216–1226 (2015)
32. Newman, M.: Networks. Oxford University Press (2018)
33. Newman, M.E., Girvan, M.: Finding and evaluating community structure in networks. Phys. Rev. E **69**(2), 026113 (2004)
34. Fortunato, S.: Community detection in graphs. Phys. Rep. **486**(3–5), 75–174 (2010)
35. Fortunato, S., Hric, D.: Community detection in networks: a user guide. Phys. Rep. **659**, 1–44 (2016)
36. Tang, L., Liu, H.: Community detection and mining in social media. Synth. Lect. Data Min. Knowl. Discov. **2**(1), 1–137 (2010)
37. Brandes, U., et al.: On finding graph clusterings with maximum modularity. In: International workshop on graph-theoretic concepts in computer science, Springer, pp. 121–132 (2007)
38. Bui, T.N., Jones, C.: Finding good approximate vertex and edge partitions is NP-hard. Inf. Process. Lett. **42**(3), 153–159 (1992)
39. Garey, M.R., Johnson, D.S., Stockmeyer, L.: Some simplified NP-complete problems. In: Proceedings of the sixth annual ACM symposium on theory of computing, pp. 47–63 (1974)
40. Park, Y., Song, M.: A genetic algorithm for clustering problems. In: Proceedings of the third annual conference on genetic programming, pp. 568–575 (1998)
41. Danon, L., Diaz-Guilera, A., Duch, J., Arenas, A.: Comparing community structure identification. J. Stat. Mech: Theory Exp. **2005**(09), P09008 (2005)
42. Lancichinetti, A., Fortunato, S., Radicchi, F.: Benchmark graphs for testing community detection algorithms. Phys. Rev. E **78**(4), 046110 (2008)