Knowledge-Based Systems 139 (2018) 149–169

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Open asynchronous dynamic cellular learning automata and its application to allocation hub location problem

Ali Mohammad Saghiri*, Mohammad Reza Meybodi

Soft Computing Laboratory, Computer Engineering and Information Technology Department, Amirkabir University of Technology (Tehran Polytechnic), 424 Hafez Ave., Tehran, Iran

ARTICLE INFO

Article history:
Received 16 June 2017
Revised 21 September 2017
Accepted 18 October 2017
Available online 20 October 2017

Keywords:
Learning automata
Cellular learning automata
Allocation hub location problem
Imprecise distances

ABSTRACT

Cellular learning automata (CLAs) are learning models that bring together the computational power of *cellular automata* and also the learning capability of *learning automata* in unknown environments. *CLAs* can be open or closed. In a closed *CLA*, the action of each *learning automaton* depends on the neighboring cells, whereas in an open *CLA*, the action of each *learning automaton* depends on the neighboring cells, and a global environment. These models can be synchronous or asynchronous. In a synchronous *CLA*, all cells are activated at the same time, but in an asynchronous *CLA*, at a given time only some cells are activated. These models can be also static or dynamic. In a dynamic *CLA*, one of its aspects such as structure, local rule or neighborhood may vary with time. All existing dynamic models of the *CLAs* are closed. In this paper, an open asynchronous dynamic *CLA* has been introduced. In order to show the potential of this model, an algorithm based on this model for solving allocation hub location problem with imprecise distances among nodes has been designed. To evaluate the proposed algorithm computer simulations have been conducted. The results of simulations show that the proposed algorithm is more robust to imprecise distances as compared to existing algorithms.

© 2017 Elsevier B.V. All rights reserved.

1. Introduction

Learning automata (*LAs*) are reinforcement learning models for decision making in unknown environments. The relationship between an *LA* and its environment is shown in Fig. 1. This model tries to find an appropriate action through repeated interaction with the environment. A single *LA* is able to solve simple problems but network of *LAs* are able to solve a wide range of problems [1].

Cellular learning automata (*CLAs*) are obtained from the combination of *Cellular Automata* (*CAs*) and *learning automata* (*LAs*) [2]. *CAs* are models which composed of a large number of cells that arranged into a lattice. In a *CA*, each cell selects a state from a finite set of states. The new state of each cell depends on the previous states of a set of cells, including the cell itself, and its neighbors [3]. In a *CLA*, each cell of the *CA* is equipped with one *LA* to select its state. In the *CLA*, a rule called local rule is used to determine the response of the *LA* of each cell. Each cell utilizes the local rule to generate the response of its *LA* considering the information about itself and its neighbors. *CLAs* have been used in wide range of applications such as computer networks [4–11], social networks [12], evolutionary computing [13,14], edge detection [15], and optimization [16]. Different models of *CLAs* are reported in the literature. Reported models for *CLAs* can be classified into two classes as described below.

- Static *CLAs (SCLAs)*: In this class, the cellular structure of the *CLA* remains fixed during the evolution of the *CLA*. Many of *CLAs* such as those reported in [6,12,16–19] are static. *SCLAs* can be either closed or open. In closed *SCLAs*, the action of each *LA* depends on the neighboring cells, whereas in open *SCLAs*, the action of each *LA* depends on the neighboring cells, a global environment, and an exclusive environment. *SCLAs* can be either synchronous or asynchronous. In a synchronous *SCLA*, all cells use their local rules at the same time [17]. This model assumes that there is an external clock which triggers synchronous events for the cells. In an asynchronous *SCLA*, at a given time only some cells are activated and the state of the rest of cells remains unchanged [18]. In [6], a model of *SCLA* with multiple *LAs* in each cell was reported. *SCLA* depending on its structure can be also classified as regular [2] or irregular [19]. In an Irregular *CLA*, the structure regularity assumption is removed. This model has been used in solving problems which can be modeled by graphs with irregular structures such as [19–22]

* Corresponding author.
E-mail addresses: a_m_saghiri@aut.ac.ir, saghiri@aut.ac.ir (A.M. Saghiri), mmeybodi@aut.ac.ir (M.R. Meybodi).

https://doi.org/10.1016/j.knosys.2017.10.021
0950-7051/© 2017 Elsevier B.V. All rights reserved.

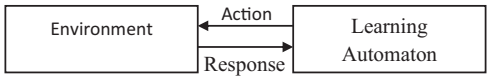

Fig. 1. Learning automaton (LA).

Table 1
Notations.

Notation	Definition
n	Number of nodes
h	Number of hubs
w_{ij}	Weight representing the amount of flow from node i to node j.
λ_c	Collection cost(per unit flow and unit distance)from an origin to a hub.
λ_d	Distribution cost (per unit flow and unit distance) from a hub to a destination.
λ_t	Transfer cost (per unit flow and unit distance) between a pair of hubs
μ_{ik}	Distance from node i to node k.
e_j	Fixed cost of establishing a hub at node 'j'
y_{ik}	y_{ik} represents assignment of spokes to hubs for each $i \neq k$, that is y_{ik} takes the value one if node i is assigned to hub k, zero otherwise. When $i = k$, $y_{kk} = 1$ means that the node is selected as hub.

- Dynamic *CLA* (*DCLAs*): In this class, one of aspects of the *CLA* such as structure, local rule or neighborhood may vary with time. *DCLAs* can be also classified as closed *DCLAs* [8,7,11,23] or open *DCLAs*. *DCLAs* can be classified as synchronous *DCLAs* or asynchronous *DCLAs*. All the reported *DCLAs* are closed and asynchronous [8,7,11,23].

The *allocation hub location problem* is an NP-hard problem [24]. In this problem, some products must be transported from a set of sources to a set of destinations. Every source or destination has a position in a plane. In this problem, a fully connected graph of all nodes containing sources and destinations is given as a problem graph. The goal of an algorithm for *allocation hub location problem* is to find a solution graph and also label the nodes to hub or spoke. The set of hub nodes is chosen to locate hubs and will be used as distribution centers for products. The other nodes are called spoke nodes. The *single allocation hub locating problem* is a version of *allocation hub location problem* [25,24]. In *single allocation hub locating problem*, spoke nodes may be allocated only to one hub. In *uncapacitated single allocation hub locating problem*, there are no capacities on the number of links of the hub nodes. This problem is known in different applications, such as telecommunications, airline industry, and computer networks [26–28]. Neural networks, tabu search, simulated annealing, and genetic algorithms are used to solve this problem in [29–34]. Before we formulate this problem the required notations are given (Table 1).

Each route can be represented by a 4-tuple, (*i; k; m; j*) where *i* and *j* represent the source and destination respectively and *k* and *m* represent the first and the second hub on the route. The aim of any *USA-HLP* algorithm is to minimize (1) considering constraints (2)–(4).

$$\begin{aligned}\min f(y) = &\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}\sum_{k=1}^{n}\lambda_c\mu_{ik}y_{ik} \\ &+ \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ji}\sum_{l=1}^{n}\lambda_d\mu_{jl}y_{jl} \\ &+ \sum_{i=1}^{n}\sum_{k=1}^{n} y_{ik}\sum_{j=1}^{n}\sum_{l=1}^{n}\lambda_t\mu_{kl}y_{jl}w_{ij} \\ &+ \sum_{j=1}^{n} y_{jj}e_j \end{aligned} \quad (1)$$

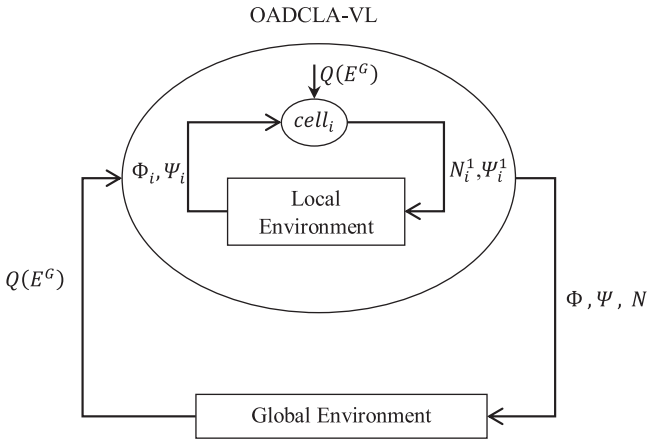

Fig. 2. The interaction of CLA with its local and global environments.

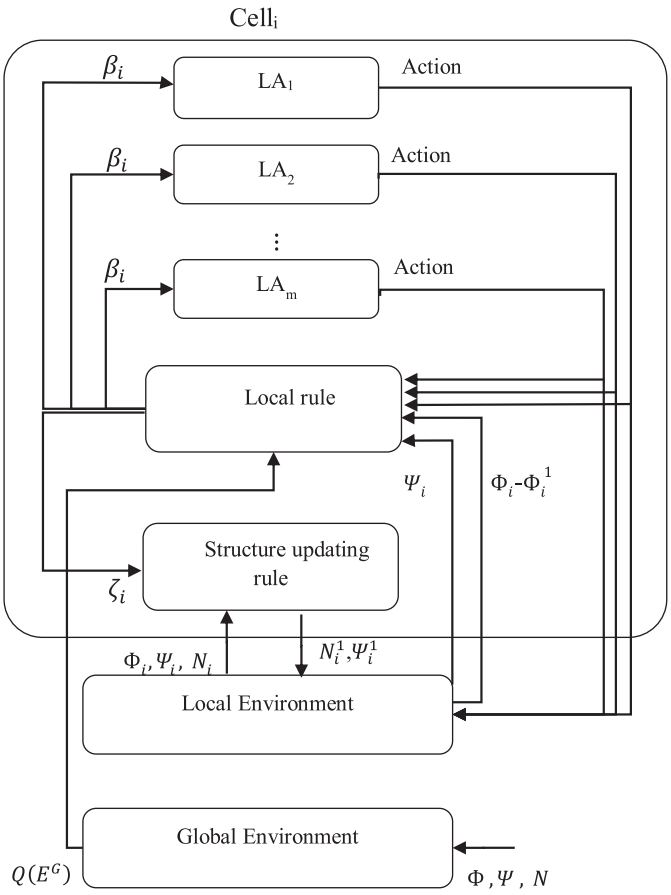

Fig. 3. Internal structure of $cell_i$ and its interaction with local and global environments.

$$\text{subject to } \sum_{k=1}^{n} y_{ik} = 1 \quad \forall_i \quad (2)$$

$$y_{kk} - y_{ik} \geq 0 \ \forall_{i,k} \quad (3)$$

$$y_{ik} \in \{0, 1\} \ \forall_{i,k} \quad (4)$$

where $f(y)$ given in (1) which we call it solution cost is the sum of the costs of collection, transfer, distribution, and the costs of the hubs for a solution. $\min f(y)$ is the cost of the optimal solution. The constraint (2) guarantees that each node is assigned to exactly one hub. The constraint (3) ensures $node_i$ is assigned to $node_k$ only if $node_k$ is a *hub*. It should be noted that the mathematical formulation used in this part is also used in [32].

Algorithm mainloop()

Begin

Repeat

Choose a cell for activation;// the mechanism of choosing the cells is application dependent//

Activate the selected cell and perform the following phases;

- **Preparation phase:** In this phase, the cell performs the following steps.
  - The cell set its attribute.
  - The cell and its neighboring cells compute their *restructuring signals* using the *local rule* (F_1).
- **Structure updating phase:** In this phase, the cell performs the following steps.
  - The neighborhood structure of the cell is updated using the structure updating rule (F_2) if the value of the *restructuring signal* of that cell is 1.
- **State updating phase:** In this phase, the cell performs the following steps.
  - Each *LA* of the cell selects one of its actions. The set of actions selected by the set of *LAs* in the cell determines the new state for that cell.
  - The local rule (F_1) is applied and a *reinforcement signal* is generated
  - The probability vectors of the *LAs* of the cell are updated.

Until (there is no cell for activation**)**

End

Fig. 4. The main loop for the operation of *DCLA*.

Begin

Repeat

Choose a $cell_{z_t}$ for activation; // The sequence according to which the cells are activated is application dependent//

Call *Activate* ($cell_{z_t}$); // Fig. 6 is the pseudo code for algorithm Activate//

Until (there is no cell to be activated**)**

End

Fig. 5. The pseudo code for the operation of *DCLA*.

An important issue when dealing with real-world applications is data imprecision. The data imprecision leads to uncertainty in decisions which should be made to solve the *uncapacitated single allocation hub locating problem*. Data imprecision has been considered to define new versions of *uncapacitated single allocation hub locating problem* which some of them are described as bellow.

- *uncapacitated single allocation hub locating problem considering imprecise demand* [26,35] (*USA-HL-IDE*).
- *uncapacitated single allocation hub locating problem considering imprecise travel time* (*USA-HL-ITT*) [36].
- *uncapacitated single allocation hub locating problem considering imprecise distance (USA-HL-IDI)* [27].
- *uncapacitated single allocation hub locating problem considering imprecise availability (USA-HL-IAV)* [28].

In *USA-HL-IDI*, the information about distances among nodes cannot be gathered with high precision which as result leads to the selection of inappropriate hub nodes. The *USA-HL-IDI* is similar to landmark selection problem reported in [10]. For both *USA-HL-IDI* and landmark selection problems, the distance among nodes cannot be calculated with high precision. The only reported algorithm for solving *USA-HL-IDI* is based on fuzzy logic [27]. In domain of solutions for landmark selection problem reported in [10], a function called *distance_evaluator* is reported to resolve the im-

Algorithm Activate ()

Input

$cell_i$

Notations

F_1 denotes the *local rule*

F_2 denotes the *structure updating rule*

Ψ_i denotes the attribute of $cell_i$

Φ_i denotes the state of $cell_i$

ζ_i denotes the restructuring signal of $cell_i$

N_i denotes the set of neighbors of $cell_i$

β_i denotes the reinforcement signal of the learning automata of $cell_i$

Begin

\\ **preparation phase**\\

Set the attribute of the $cell_i$.

Compute ζ_i using F_1;

Ask from neighboring cells of $cell_i$ to compute their *restructuring signals*;

Gather the *restructuring signals* of the Neighboring cells;

\\ **structure updating phase**\\

If (ζ_i is 1) **Then**

Compute N_i and Ψ_i using F_2;

EndIf

state updating phase\\

Each *learning automaton* of $cell_i$ chooses one of its actions;

Set Φ_i;// set Φ_i to be the set of actions chosen by the set of *learning automata* in $cell_i$

Compute β_i using F_1;

Update the action probabilities of *learning automata* of $cell_i$ using β_i;

End

Fig. 6. The pseudo code of the process which is executed upon the activation of a cell.

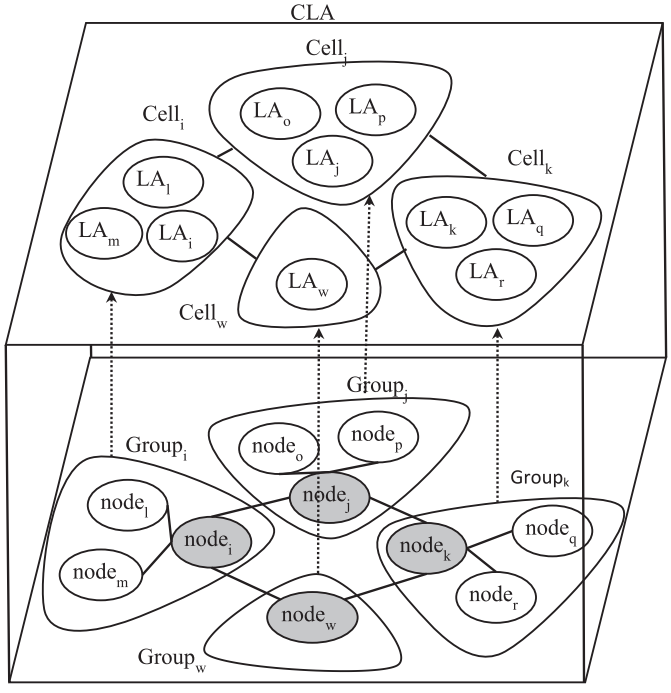

Fig. 7. A snapshot of the CLA.

```
Algorithm Shuffle()
Input
        A  // A vector of n items
        n  // number of items of A
Output
        A  // A random permutation of n items
Begin
  For each i=0,…, n-1 do
    Choose integer j ( i<j<n ) uniformly at random;

    Swap A[i] and A[j];
  EndFor
End
```

Fig. 8. Knuth shuffle [48].

```
Algorithm ALLOCATE-CLA()
Notations
        z_t is the index of the cell activated in iteration t
        n is the number of the cells
Input
        k  // order of the activation sequence
Begin
  Generate an activation sequence with order k; // Algorithm shuffle
                given in Fig.  8 is used  to generate the
                  activation sequence//
  t←0;
  Repeat
    t←t+1;
    Call Activate(cell_{z_t}) ;// Algorithm activate given in Fig.6
                  is  used to activate the cell//
    Set the label of the nodes according to the states of their
                          corresponding  cells;
    Save the information of the structure of the CLA;
  Until (t<n*k)
End
```

Fig. 9. The pseudo code for *ALLOCATE-CLA*.

Fig. 10. Input and output of the global environment.

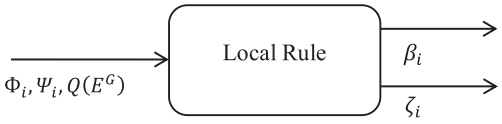

Fig. 11. Input and output of the local environment of $cell_i$.

precision embedded in the distances among nodes. Function *distance_ evaluator* takes two values (x and y), and threshold $r \in [0, 1)$ as input and then return true if $(x \geq y \text{ and } (1-r) \times x \leq y)$**or** $(x < y \text{ and } (1-r) \times y \leq x)$ and false otherwise. This function determines whether two values can be considered equal or not based on parameter r. we will use this function to propose a new solution for *USA-HL-IDI*.

In this paper, at first, we introduce an open asynchronous dynamic model of *CLA* and then based on this model an algorithm, called *ALLOCATE-CLA*, for solving the allocation hub location problem with imprecise distances is proposed. To show the efficiency of the proposed algorithm computer experimentations have been conducted and the results are compared with the results obtained for *Zarandi's algorithm* [27] which is the only reported algorithm for solving this problem. The results show the superiority of the proposed algorithm. The results also show that using *CLA* gives the algorithm the ability of removing the imprecision embedded in the distances among nodes while trying to find the solution.

```
Algorithm Node_Categorizer
Input
        group_i // the set of nodes of cell_i//
        r  // a threshold//
        u  // a vector containing distances from all nodes of group_i to
                the hub nodes of the groups adjacent to group_i //
Output
        c-node  // a candidate node//
        d-nodes  // set of departed nodes//
        m  // mean of distances computed for node c-node//
        v  // variance of distances computed for node  c-node//
Begin
  Find the c-node; // a c-node (candidate node) is a node that has the
                minimum mean and variance of
                distances to the hub nodes of the
                groups adjacent to group_i and spoke
                nodes of group_i //
  Find the d-nodes; // In group_i, a node_d is a d-nodes (departed node)
                if the distances from node_d to the hub
                nodes of the adjacent groups of  group_i
                 is not equal (based on the output of
                function distance_ evaluator  given
                in Fig.  13) to the distances from the
                c-node to the hub nodes of the adjacent
                groups of group_i //

  Return (c-node, d-node, m, v);
End
```

Fig. 12. The pseudo code for function *Node_Categorizer*.

```
Algorithm distance_ evaluator
Input
        r // a threshold//
        x // a  real value//
        y // a real value//
Output
        z // a value which determines whether two values x and y
          can be considered equal or not considering parameter r//
Begin
  If (x ≥ y and (1 − r) × x ≤ y)or
    (x < y and (1 − r) × y ≤ x) Then
      z ← True;
  Else
      z ← False;
  EndIf
  Return (z);
End
```

Fig. 13. The pseudo code for function *distance_ evaluator*.

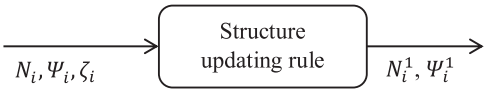

Fig. 14. Input and output of the structure updater of $cell_i$.

The rest of this paper is organized as follows. Section 2 provides basic information about the learning automata. The proposed model of the *CLA* is given in Section 3. In Section 4, an algorithm for allocation hub location problem utilizing the proposed model is introduced. Section 5 reports the experimental results and Section 6 concludes the paper.

2. Learning automata

In this section, the *LAs* and their learning algorithms are explained. The learning process of an *LA* is described as follows. The

Algorithm Structure updating rule
Input
The neighbors, attributes and restructuring signals of $cell_i$ and its neighbors
Output
Immediate neighbors and attribute of $cell_i$
Notations
Let $group_i$ be the set nodes of $cell_i$.
Let $group_j$ be the set nodes of $cell_j$.
Let d-nodes be the set of departed nodes of $cell_i$.
Let c-nodes be the candidate node of $cell_i$.

```
Begin
 If (the restructuring signal of cell_i is equal to 1) Then
  Find set d-nodes;// using function Node_categorizer which is given in
                   Fig. 12//
  If (d-nodes is not empty) Then
    Select at random a node node_k from set d-nodes;
    Select an appropriate group group_j for node_k;
    If (group_j has been found) Then
      If (group_i has one node) Then
        Call Join;
      Else
        Call Transfer;
      EndIf
    Else
      Call Shrink;
    EndIf
  EndIf
 EndIf
End
```

Fig. 15. The pseudo code of the structure updater.

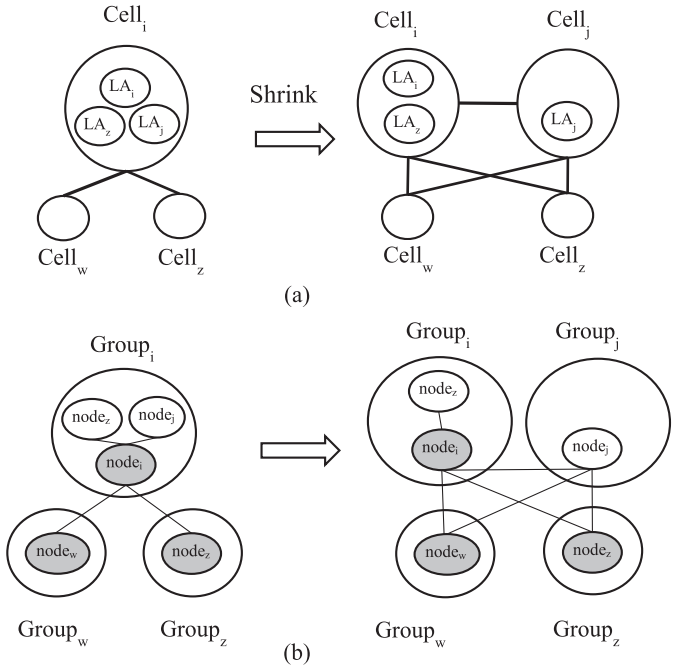

Fig. 17. (a) An example of *shrink* operation. (b) The effect of *shrink* operation on the structure of the groups of the proposed algorithm.

LA randomly selects an action from its action set and then performs it on the environment. The environment then evaluates the chosen action and responds with a reinforcement signal (reward or penalty) to the *LA*. According to the reinforcement signal of the environment to the selected action, the *LA* updates its action probability vector and then the learning process is repeated. The updating algorithm for the action probability vector is called the learning algorithm. The aim of the learning algorithm of the *LA* is to find an appropriate action from the set of actions so that the average reward received from the environment is maximized. The *LAs* can be classified into two classes, fixed and variable structure *LAs* [1,37]. Variable structure *LAs* which is used in this paper is represented by quadruple $\langle \underline{\beta}, \underline{\alpha}, \underline{P}, G \rangle$, where $\underline{\beta}$ a set of inputs actions (called response or reinforcement signal), $\underline{\alpha}$ a set of outputs, P denotes the state probability vector, and G is learning algorithm. The learning algorithm is used to modify the probability vector.

$$\begin{aligned} p_i(k+1) &= p_i(k) + a(1 - p_i(k)) \\ p_j(k+1) &= p_j(k) - a p_j(k), \quad \forall j \neq i \end{aligned} \tag{5}$$

$$\begin{aligned} p_i(k+1) &= (1-b)p_i(k) \\ p_j(k+1) &= \frac{b}{r-1} + (1-b)p_j(k), \quad \forall j \neq i \end{aligned} \tag{6}$$

Let α_i be the action chosen at step k as a sample realization from distribution $P(k)$. In a linear learning algorithm, equation for updating probability vector $P(k)$ is defined by (5) for a favorable response ($\beta = 1$), and (6) for an unfavorable response ($\beta = 0$). Two

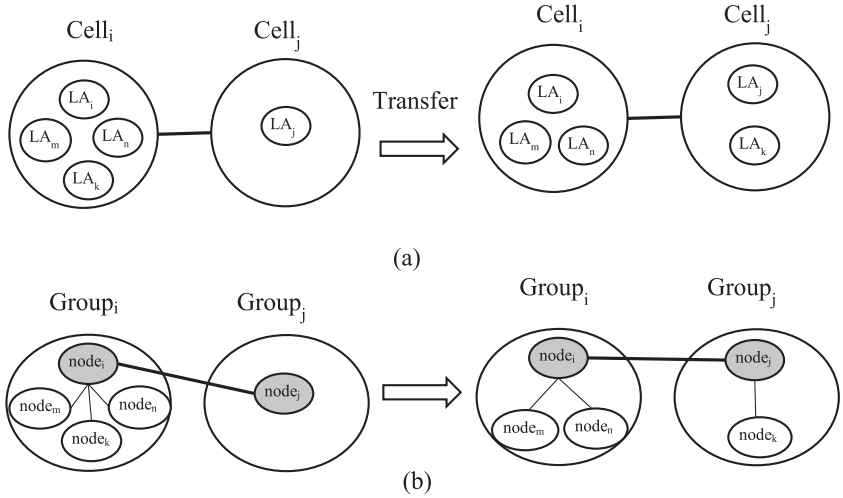

Fig. 16. (a) An example of *transfer* operation for two cells. (b) The impact of *transfer* operation on the structure of the groups.

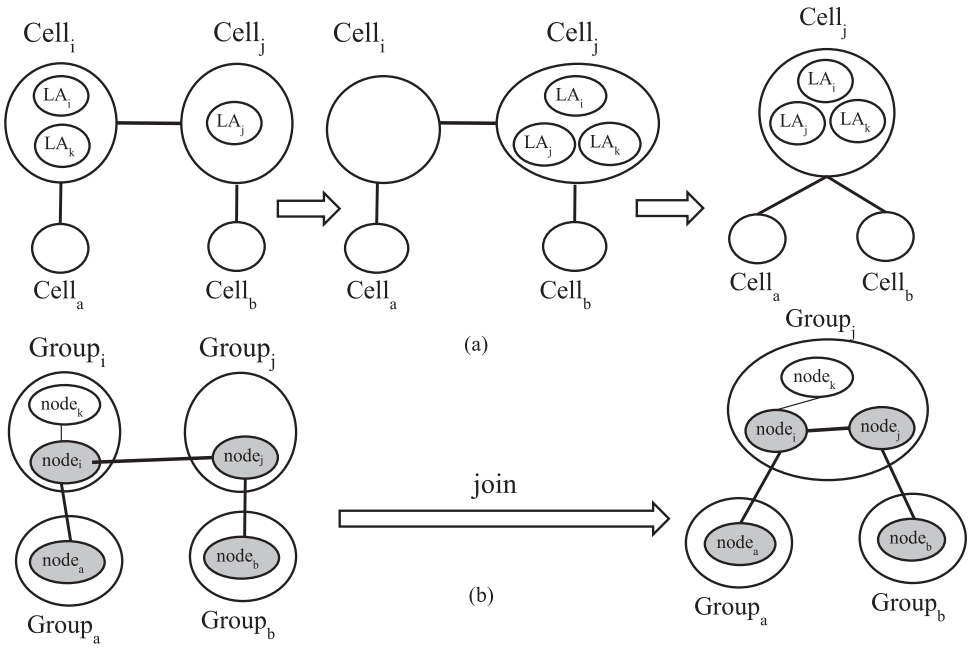

Fig. 18. (a) An example of *join* operation. (b) The impact of *join* operation on the structure of the groups.

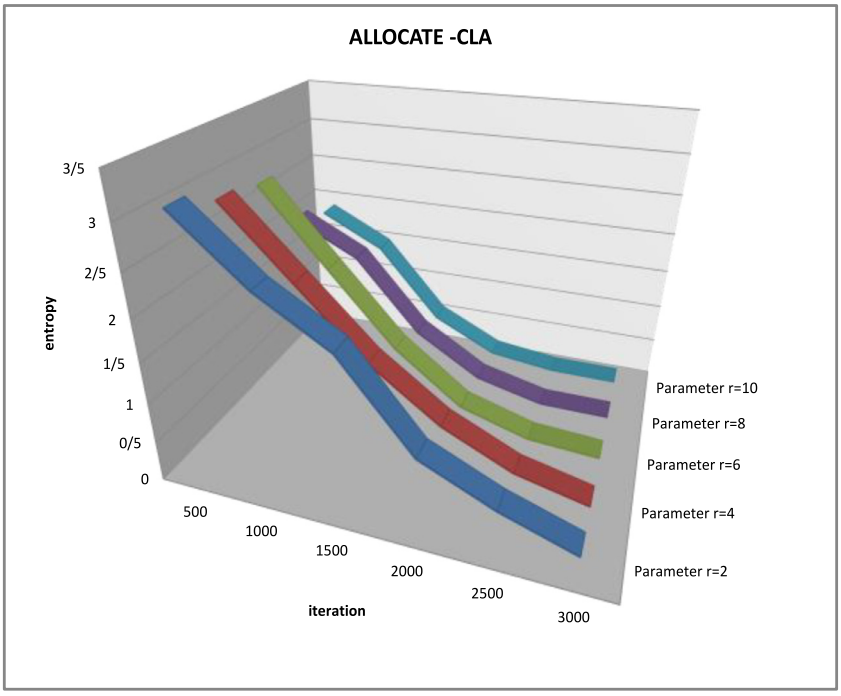

Fig. 19. The impact of parameter r on *entropy* of *ALLOCATE-CLA* during its operation.

parameters *a* and *b* represent reward and penalty parameters, respectively. The parameter *a* (*b*) determines the amount of increase (decreases) of the action probabilities. *r* denotes the number of actions that can be taken by the *LA*. If a = b, the above learning algorithm is called linear reward penalty (L_{RP}); if a $\gg$ b the learning algorithm is called linear reward-ε penalty ($L_{R\varepsilon P}$); and finally if b = 0, it is called linear reward inaction (L_{RI}) algorithm.

In the recent years, theory of learning automata has been arisen to different applications such as function optimization [38], Grid computing [39], social networks [40], stochastic graphs [41,42], Petri nets [43], computer vision [44], differential evolution [45], and complex networks [46,47], to mention a few.

3. Open asynchronous dynamic cellular learning automaton with varying number of learning automata in each cell (OADCLA-VL)

In this section, at first, a definition of *OADCLA-VL* is given, then its updating process is explained, and finally to study the updating process two metrics are introduced

3.1. Definition of OADCLA

An *OADCLA-VL* is a network of cells whose structure changes with time and each cell contains a varying set of *LAs* and a set

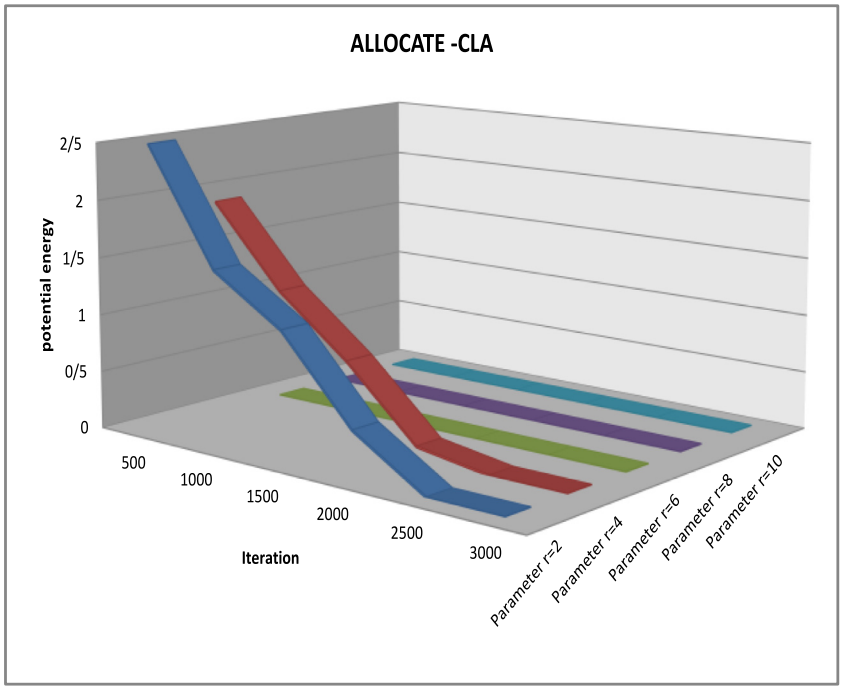

Fig. 20. The impact of parameter r on *potential energy* of *ALLOCATE-CLA* during its operation.

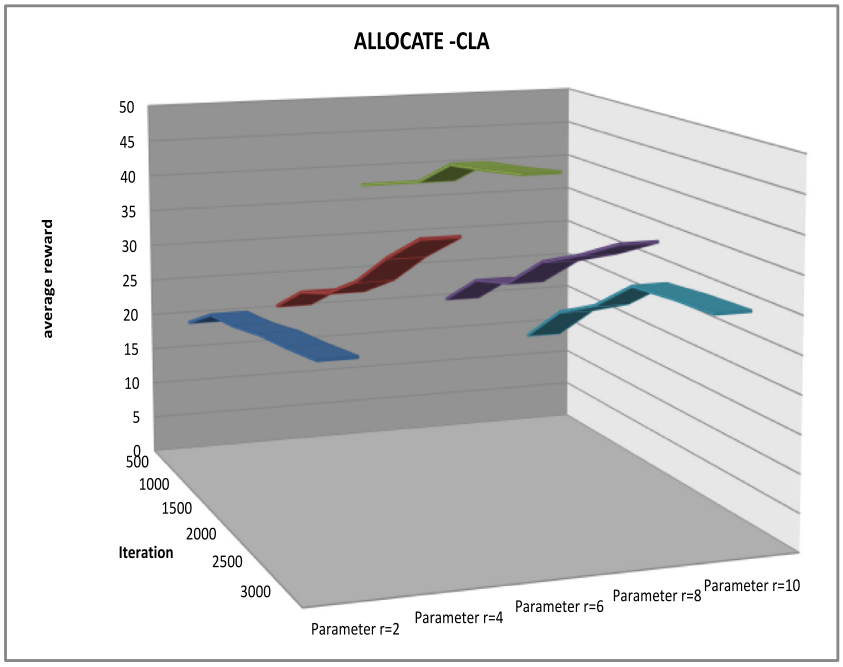

Fig. 21. The impact of parameter r on *average reward* of *ALLOCATE-CLA* during its operation.

of attributes. In this model the action of each *LA* depends on the neighboring cells, and a global environment. This model can be formally defined by 8 tuples as follows

$$OADCLA - VL = (G, A, \Psi, \Phi, N, E^G, F_1, F_2)$$

where:

- $G = (V, E)$ is an undirected graph which determines the structure of *OADCLA-VL* where $V = \{cell_1, cell_2, \ldots, cell_n\}$ is the set of vertices and E is the set of edges. $A = \{LA_1, LA_2, \ldots, LA_v\}$ is a set of *LAs*. A subset of set A is assigned to a cell.
- $N = \{N_1, N_2, \ldots, N_n\}$ where $N_i = \{cell_j \in V | dist(cell_i, cell_j) < \theta_i\}$ where θ_i is the neighborhood radius of $cell_i$ and $dist(cell_i, cell_j)$ is the length of the shortest path between $cell_i$ and $cell_j$ in G. N_i^1 determines the immediate neighbors of $cell_i$.
- $\Psi = \{\Psi_1, \Psi_2, \ldots, \Psi_n\}$, $\Psi_i = \{(j, X_j, C_j) | \ cell_j \in N_i\}$ denotes the attribute of $cell_i$ where $X_j \subseteq \{x_1, x_2, \ldots, x_s\}$ and $C_j \subseteq A$. $\{x_1, x_2, \ldots, x_s\}$ is the set of allowable attributes. Ψ_i^1 determines the attribute of $cell_i$ when $\theta_i = 1$. Ψ_i^l determines the attribute of $cell_i$ when $\theta_i = l$.
- $\Phi = \{\Phi_1, \Phi_2, \ldots, \Phi_n\}$ where $\Phi_i = \{(j, k, \alpha) | cell_j \in N_i$ *and action* α_l has been chosen by $LA_k \in C_i\}$ denotes the state of

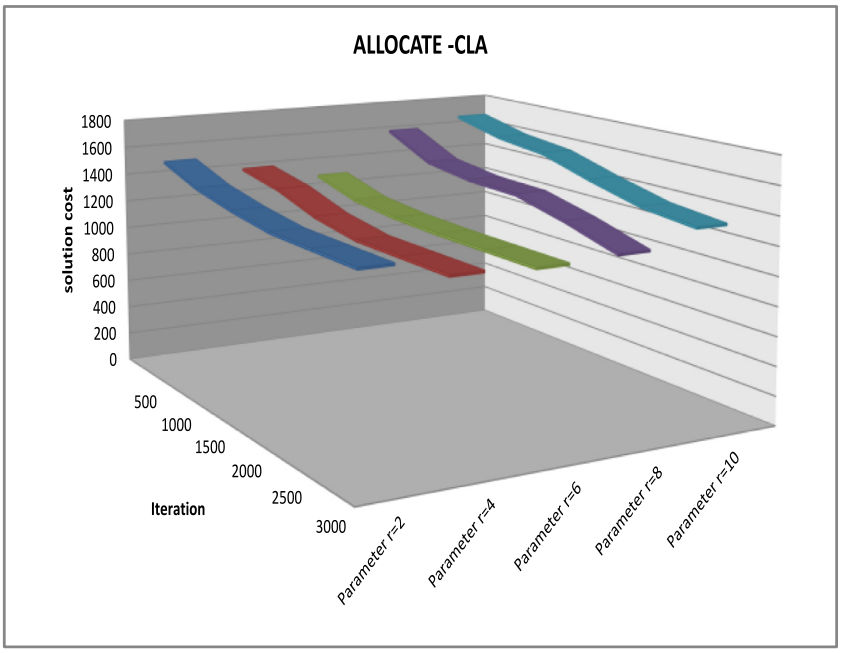

Fig. 22. The impact of parameter r on *solution cost* of *ALLOCATE-CLA* during its operation.

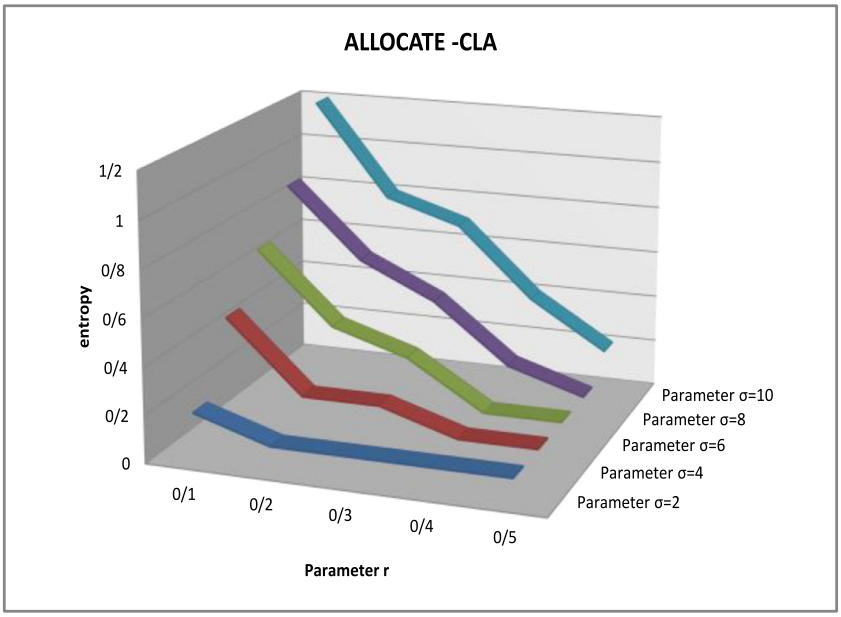

Fig. 23. The impact of parameter r on *entropy* of *ALLOCATE-CLA* at iteration 3000.

$cell_i$. Φ_i^1 determines the state of $cell_i$ when $\theta_i = 1.\Phi_i^l$ determines the state of $cell_i$ when $\theta_i = l$.

- E^G is the global environment of the *CLA*. The *CLA* operates under this environment. This environment is defined by the application. The input to the global environment is (N, Φ, Ψ) and its output is $Q(E^G)$
- $F_1 : (\underline{\Phi}, \underline{\Psi}, \underline{Q(E^G)}) \rightarrow (\underline{\beta}, \underline{\zeta})$ is the local rule of *OADCLA-VL*. In each cell, the local rule computes the reinforcement signal and the restructuring signal for the cell based on the output of the global environment, the states and attributes of that cell and its neighboring cells. for example, in $cell_i$, local rule takes $< \Phi_i, \Psi_i, Q(E^G) >$ and returns $< \beta_i, \zeta_i >$. The reinforcement signal is used to update the *LA* of that cell. Note that, ζ_i is the restructuring signal of $cell_i$.
- $F_2 : (\underline{N}, \underline{\Psi}, \underline{\zeta}) \rightarrow (\underline{N}^1, \underline{\Psi}^1)$ is the structure updating rule. In each cell, the structure updating rule finds the immediate neighbors and attribute of that cell. For example, in $cell_i$, structure updating rule takes $< N_i, \Psi_i, \zeta_i >$ and returns N_i^1 and Ψ_i^1.

It is obvious that, the environment under which a cell of the *CLA* operates consists of two parts: local environment and global environment as shown in Fig. 2. The definitions of these environments are application dependent. Fig. 3 shows the internal structure of $cell_i$ and its interaction with local and global environments.

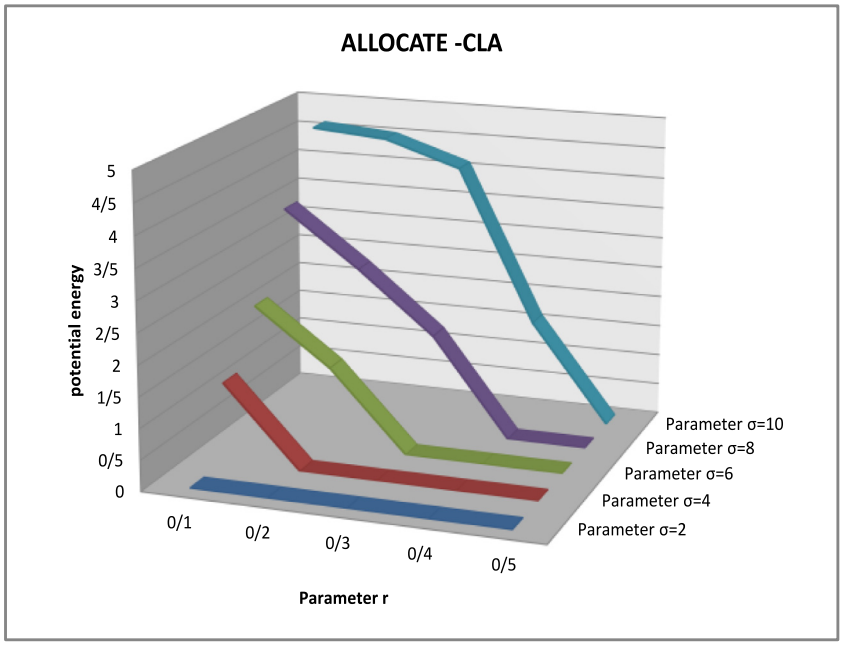

Fig. 24. The impact of parameter r on *potential energy* of *ALLOCATE-CLA* at iteration 3000.

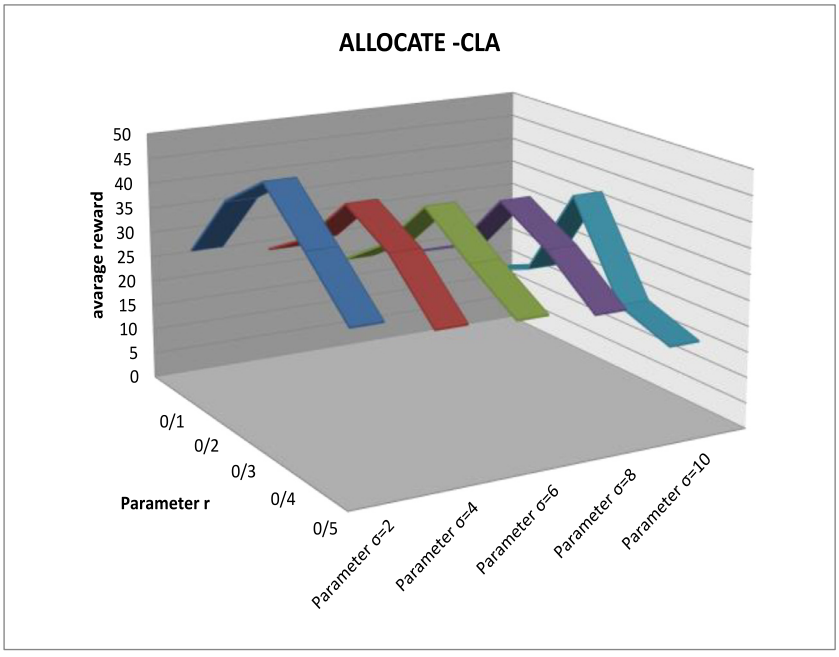

Fig. 25. The impact of parameter r on *average reward* of *ALLOCATE-CLA* at iteration 3000.

3.2. Updating process of OADCLA

The main loop for the operation of the *CLA* is described in Fig. 4. The application determines which cell must be activated. Upon activation of a cell, the cell performs a process containing three phases: **preparation, structure updating** and **state updating**.

In what follows, we give a more detailed version of the operation of *DCLA* using the notations given for *DCLA*. Fig. 4 shows the main loop describing the operation of *DCLA* and Fig. 5 shows the pseudo code of the process which is performed upon the activation of a cell of *DCLA*.

3.3. Evaluation metrics

The performance of *DCLAs* will be studied using two metrics: ***potential energy***, and ***entropy*** which are defined as below.

Potential energy: The ***potential energy*** of the *CLA* is defined by Eq. (7) given below

$$T(t) = \sum_{i=1}^{n} \zeta_i \tag{7}$$

where ζ_i (t) is the restructuring signal of $cell_i$ at iteration *t. restructuring tendency* can be used to study the changes in the structure of *CLA* as it interacts with the environment. If the value of *T(t)* becomes zero then no further change needs to be made to the struc-

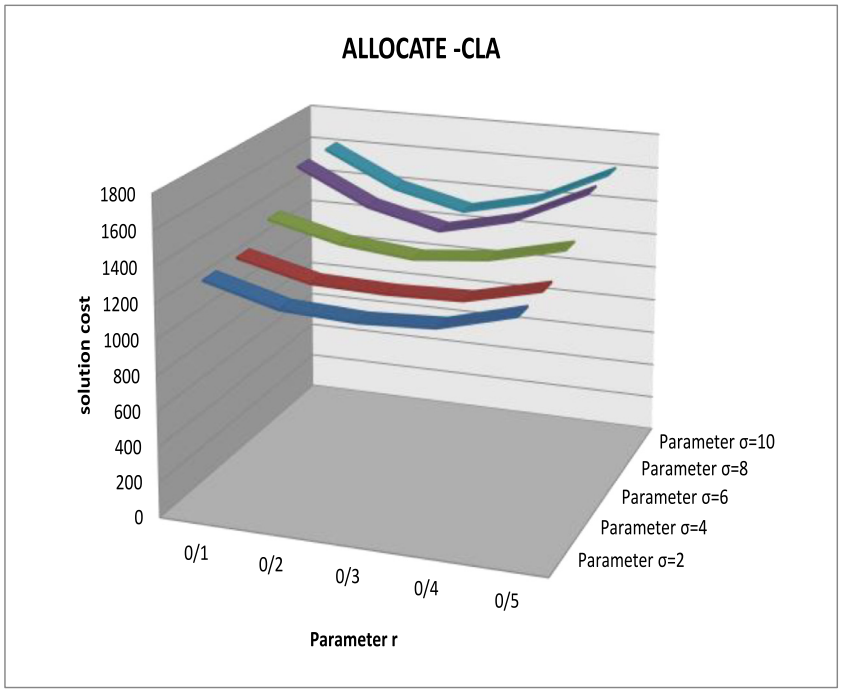

Fig. 26. The impact of parameter r on *solution cost* of *ALLOCATE-CLA* at iteration 3000.

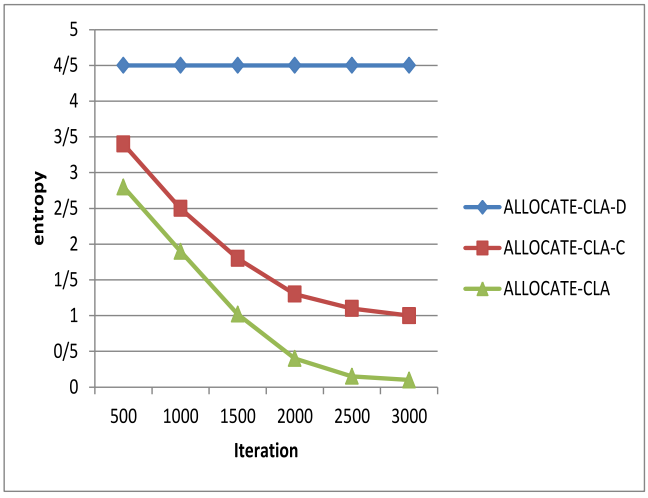

Fig. 27. Comparison of *ALLOCATE-CLA*, *ALLOCATE-CLA-C* and *ALLOCATE-CLA-D* with respect to entropy.

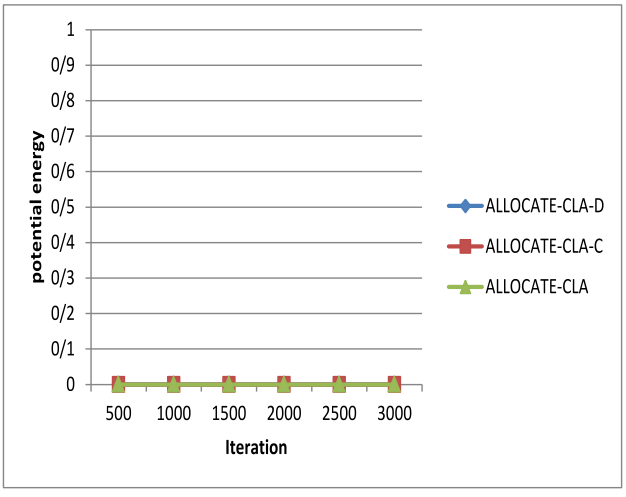

Fig. 28. Comparison of *ALLOCATE-CLA*, *ALLOCATE-CLA-C* and *ALLOCATE-CLA-D* with respect to potential energy.

ture. Higher value of $T(t)$ indicates higher disorder in the structure of *CLA*.

Entropy: The ***entropy*** of the CLA at iteration t is defined by Eq. (8) given below

$$H(t) = -\sum_{l=1}^{n} H^l(t) \tag{8}$$

where n is the number of cells of the *CLA*, $H^l(t)$ is the *entropy* of cell $cell_l$ of *CLA*. The *entropy* of cell $cell_l$ is defined by Eq. (9) given below

$$H^l(t) = \sum_{i \in C_l(t)} H_i(t) \tag{9}$$

where $H_i(t)$ is the *entropy* of learning automaton LA_i and $C_l(t)$ denotes the set of indices of learning automata of cell $cell_l$ at iteration t. $H_i(t)$ is the *entropy* of learning automaton LA_i defined Eq. (10) given below

$$H_i(t) = \sum_{j=1}^{r_i} p_{ij}(t).\ln\left(p_{ij}(t)\right) \tag{10}$$

where r_i is the number of actions of learning automaton LA_i. $p_{ij}(t)$ is the probability of selecting action α_j of learning automaton LA_i at iteration t. The value of $H(t)$ may be used to study the changes that occur in the states of the cells *of CLA*. The value of zero for $H(t)$ means that the learning automata of the cells of the *CLA* no longer change their action. Higher values of $H(t)$ mean higher rates of changes in the actions selected by learning automata residing in the cells of the *CLA* [8,9].

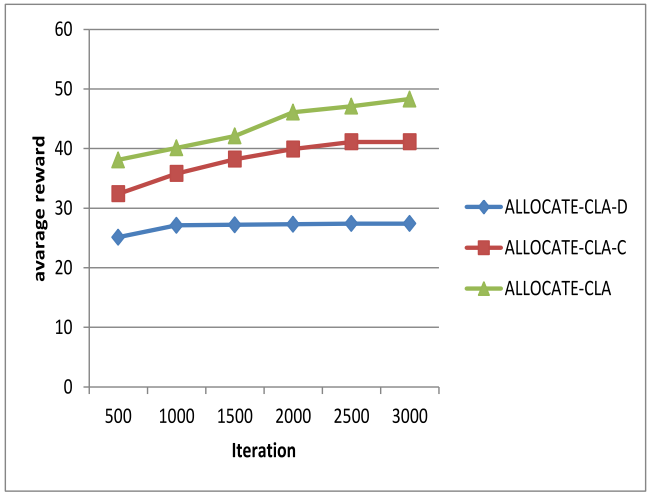

Fig. 29. Comparison of *ALLOCATE-CLA*, *ALLOCATE-CLA-C* and *ALLOCATE-CLA-D* with respect to average reward.

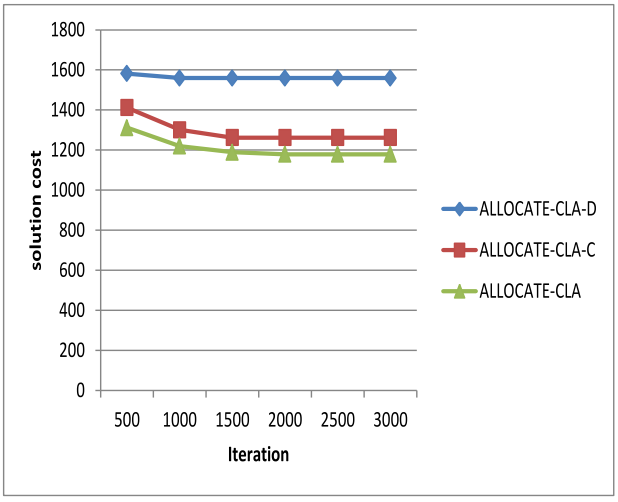

Fig. 30. Comparison of *ALLOCATE-CLA*, *ALLOCATE-CLA-C* and *ALLOCATE-CLA-D* with respect to solution cost.

4. An algorithm for hub allocation problem utilizing OADCLA-VL:ALLOCATE-CLA

Initially, a *CLA* isomorphic to the graph of the hub problem is created which involves defining the initial structure, local rule, structure updating rule, and global environment. The graph of the problem is then mapped into the *CLA* in which each cell is equipped to a *learning automaton*. A node in a cell of the *CLA* is initially labeled with "free". The *learning automata* in each cell has two actions "set the label to hub node" and "set the label to spoke node". As the Algorithm proceeds, the structure of *CLA* changes and as a results the number of nodes in a cell may change. We call the set of nodes in each cell a group of nodes. Clearly, at the beginning of the algorithms each group contains one node of the graph of the problem and hence the number of groups is equal to the number of cells in the *CLA*. Each cell of the *CLA* has an attribute which is the position of the corresponding node in the graph of problem. Fig. 7 shows a snapshot of the *CLA* at a particular time. As shown, $group_i$, $group_j$, $group_k$ and $group_w$ are the groups of $cell_i$, $cell_j$, $cell_k$, and $cell_w$, respectively and $node_i$, $node_j$, $node_k$, and $node_w$ determined by the algorithm to be the *hub* nodes of $group_i$, $group_j$, $group_k$, and $group_w$ respectively. Once the *CLA* is created and the nodes are labeled as explained above, the algorithm proceeds according to the framework described for the operation of *CLA* in the previous section. The pseudo code of the CLA based Allocation algorithm now can be described in Fig. 9. In the proposed algorithm, an activation sequence denoted by $\{z_t\}_{t=0}^{\infty}$ determines the order under which the cells are activated. z_t denotes the index of the cell activated at iteration t. The mechanism used for generating the activation sequence is described as follows. A random activation sequence with order k is composed of concatenation of k random permutation of indices of the cells. The algorithm reported in [48] known as the Knuth shuffle is used to generate the random

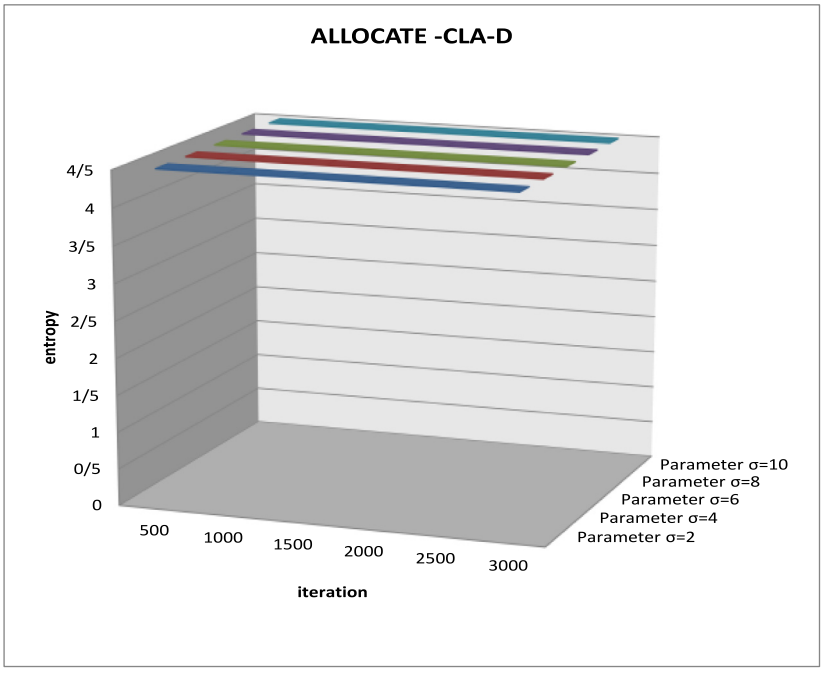

Fig. 31. The impact of σ on *entropy* of *ALLOCATE-CLA-D*.

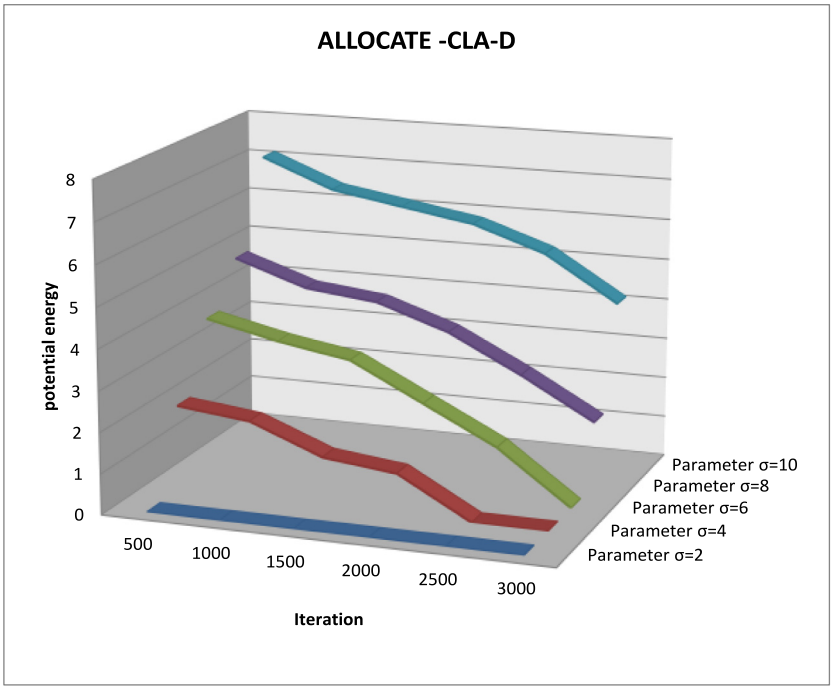

Fig. 32. The impact of σ on *potential energy* of *ALLOCATE-CLA-D*.

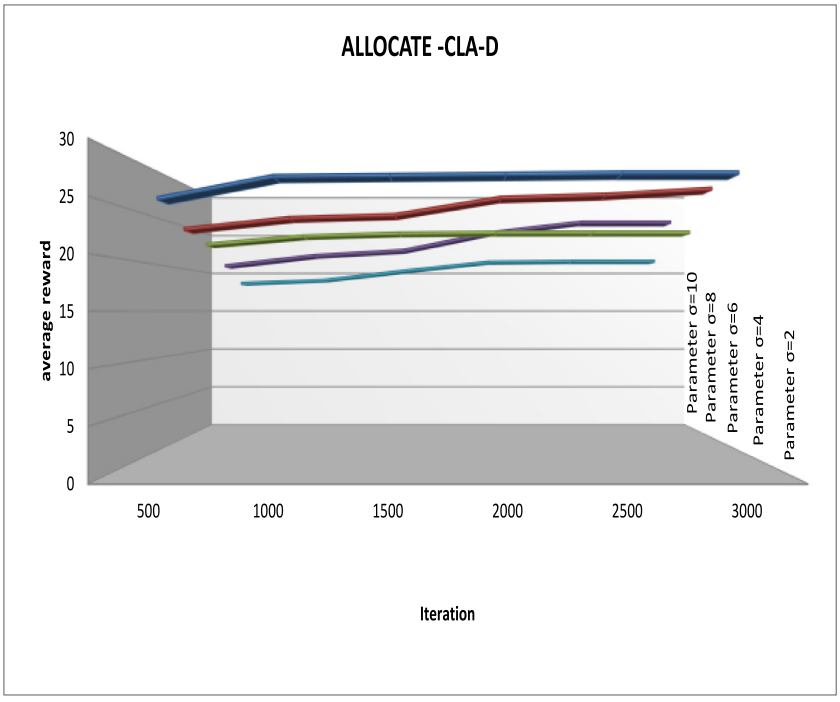

Fig. 33. The impact of σ on *average reward* of *ALLOCATE-CLA-D*.

permutation of indices. In this algorithm, any permutation of n elements will be produced with probability $1/n!$.

Remark 1. It should be noted that, the proposed algorithm will use a function called *Node_Categorizer* to organize the groups of nodes. This function is borrowed from of *Voronoi* diagram construction algorithm reported in [49]. In addition, the proposed algorithm will use a function called *distance_ evaluator* to resolve the measurement error of distance evaluation among nodes. In each group, a team of *LAs* are organized for labeling the nodes.

To complete the description of the algorithm we need to describe the **Global environment, Local rule**, and **structure updating rule** for the CLA used by function *Activate*($cell_{z_t}$) called by Algorithm *ALLOCATE-CLA* as given in Fig. 9.

- **Global environment**: The input and output of this unit are shown in Fig. 10. In the global environment, the objective function (defined according to the application) is applied to the input of the environment and provides an output. If the value of the objective function (obtained using Eq. (1)) is greater than the maximum value observed in all previous iterations, then

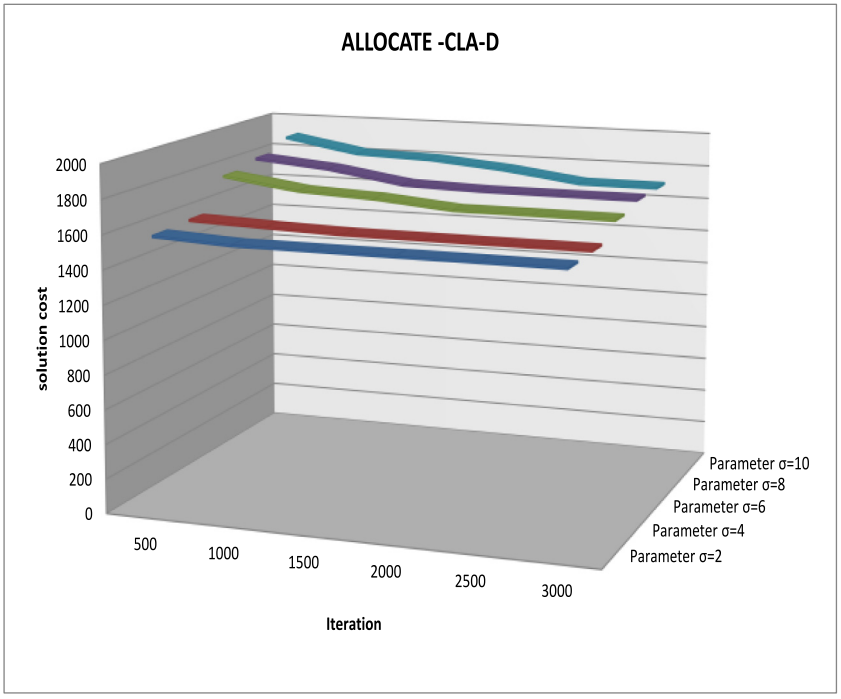

Fig. 34. The impact of σ on *solution cost* of *ALLOCATE-CLA-D*.

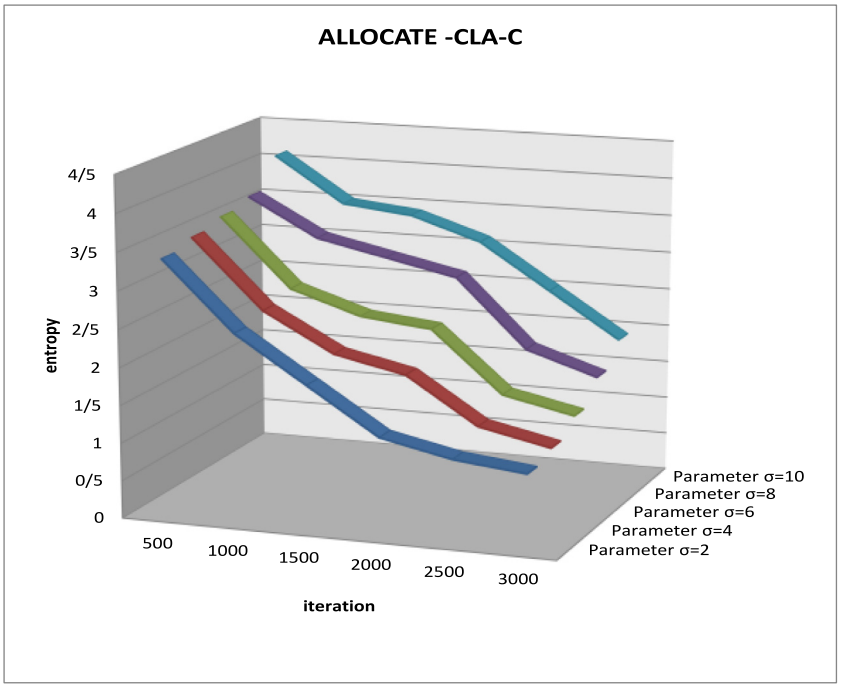

Fig. 35. The impact of σ on *entropy* of *ALLOCATE-CLA-C*.

the output of the global environment is "Explore"‖ and is "Exploit" otherwise.

- **Local rule**: The input and output of local environment are shown in Fig. 11. Local rule takes information of immediate neighbors of a cell *celli* as input and then returns reinforcement signal (β_i) and restructuring signal (ζ_i) of *celli*
  - In *celli*, the restructuring signal ζ_i is set to 1 if there is at least one departed node (returned by function Node_Categorizer (Fig. 12) and returns 0 otherwise.
  - In celli, the reinforcement signal β_i is computed by an algorithm described as follows. The reinforcement signal is 1, if the LA of nodej (returned by the Node_Categorizer function as c-node) has selected "set the label to hub node", other LAs of *celli* have selected "set the label to spoke node", the output of the global environment is "Exploit", and there is no departed node (returned by the Node_Categorizer function as d-node), and the reinforcement signal is 0 otherwise. The function Node_Categorizer uses the function dis-

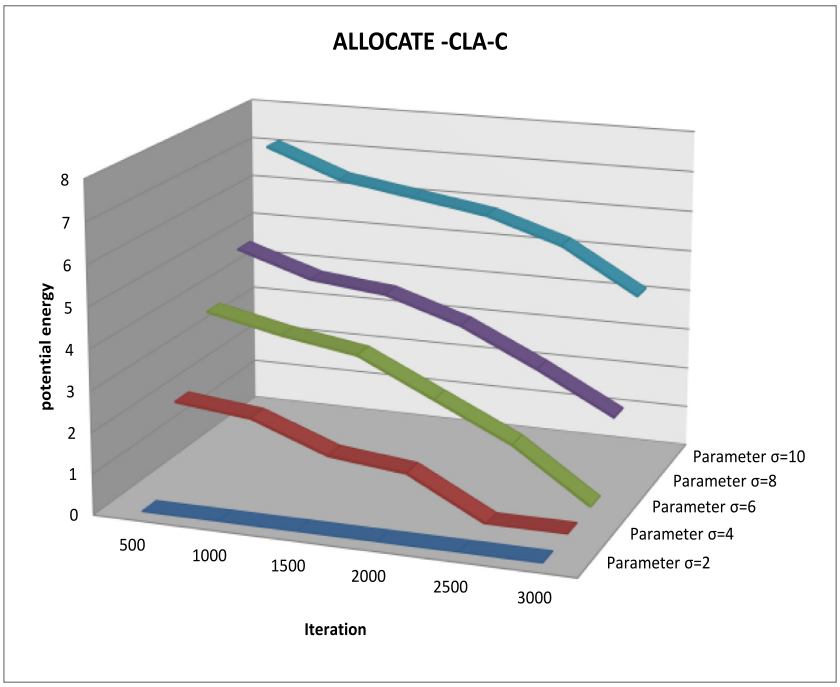

Fig. 36. The impact of σ on *potential energy* of *ALLOCATE-CLA-C*.

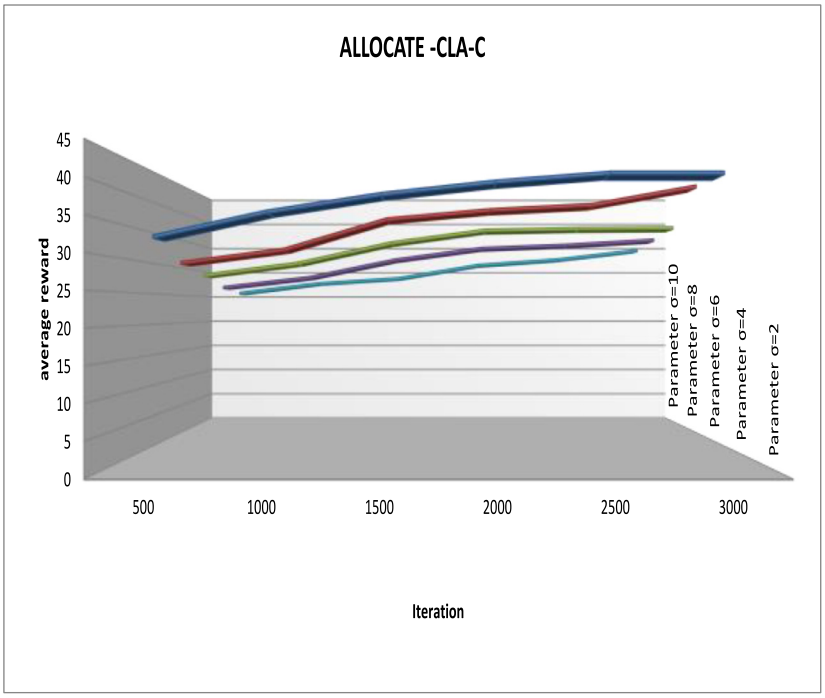

Fig. 37. The impact of σ on *average reward* of *ALLOCATE-CLA-C*.

tance_evaluator (Fig. 13) to compare the distances among nodes.

- ***Structure updating rule***

The input and output of the structure updating rule are shown in Fig. 14. This unit is implemented using three operations called ***Transfer*** operation, ***Shrink*** operation, and ***Join*** operation which are described below Fig. 15.

In what follows we describe operations ***Transfer***, ***Shrink***, and ***Join*** by examples.

1. ***Transfer operation***: Fig. 16 shows an example of the *transfer* operation. In this figure, $cell_i$ and $cell_j$ have participated in a *transfer* operation. Let $group_i$ and $group_j$ denote the groups of node for $cell_i$ and $cell_j$ respectively and $node_i$ and $node_j$ are the *hub* nodes for $group_i$ and $group_j$, respectively and also $node_k$ is returned by *Node_Categorizer* as a *departed* node for $group_i$. In $group_i$, the *structure updating rule* of $cell_i$ uses *Transfer* operation to move LA_k (the learning automaton of $node_k$) to cell $cell_j$. $cell_j$ is a neighboring cell of $cell_i$ whose hub node has the least distance to the hub node of $cell_i$.

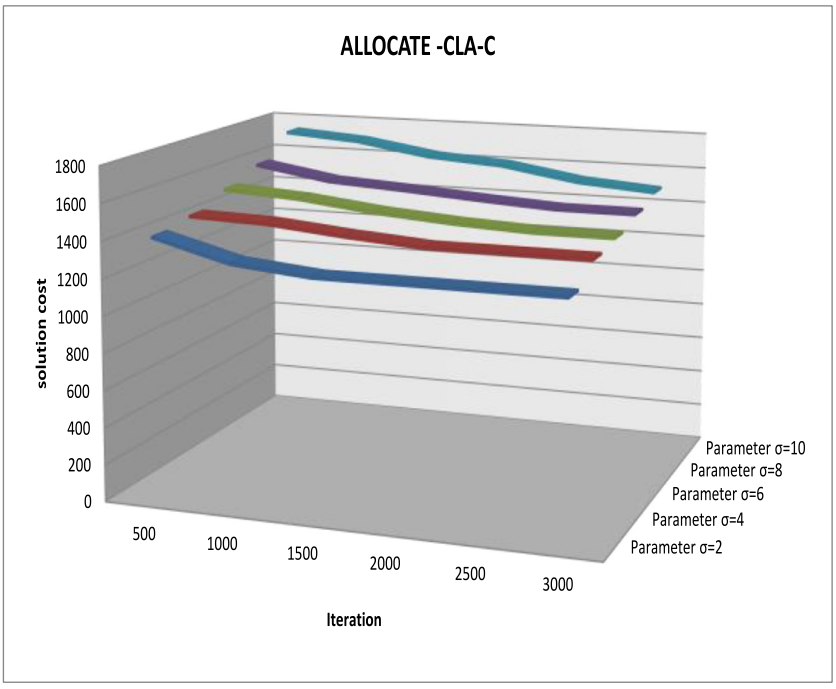

Fig. 38. The impact of σ on *solution cost* of *ALLOCATE-CLA-C*.

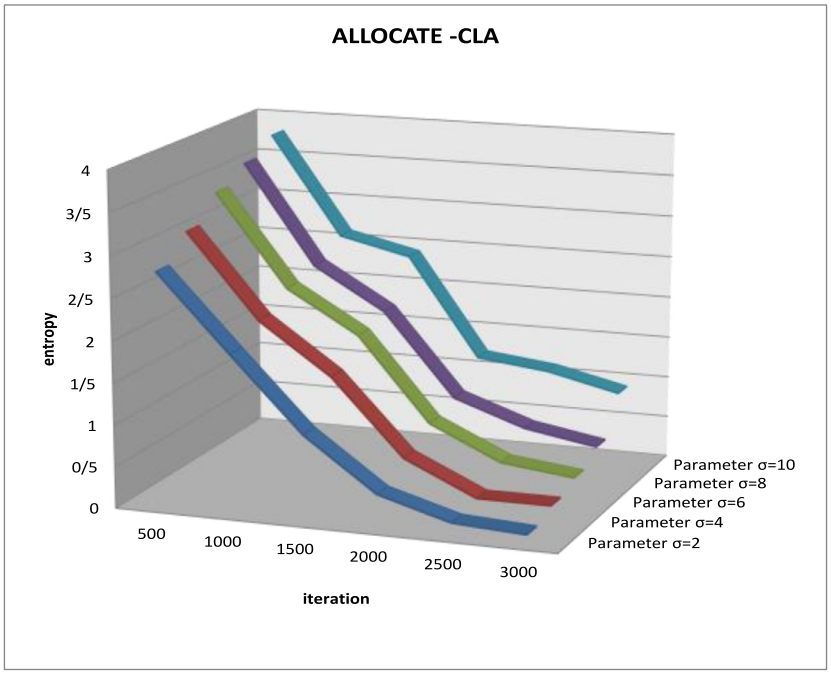

Fig. 39. The impact of σ on *entropy* of *ALLOCATE-CLA*.

2. ***Shrink operation***: Fig. 17 shows an example of the *shrink* operation. In this figure, $cell_i$, $cell_j$, $cell_w$, and $cell_z$ have participated in a *shrink* operation. Let $group_i$ and $group_j$ are the groups of node for $cell_i$ and $cell_j$, respectively, and $node_j$ is returned by *Node_Categorizer* as a *departed* node for $group_i$. In $group_i$, if $node_i$ cannot find an appropriate cell for executing the transfer operation with it, the *structure updating rule* of $cell_i$ uses *shrink* operation to split the $cell_i$ into two cells; a new cell which contains the learning automaton of $node_j$ and a cell which contains the rest of the learning automata of $cell_i$. Note that the neighbors of the new cell are the neighbors of $cell_i$ (before execution of *shrink* operation).
3. ***Join operation***: Fig. 18 shows an example of *join* operation. In this figure, $cell_a$, $cell_b$, $cell_i$ and $cell_j$ have participated in a *join* operation. The *join* operation is performed if after the execution of *transfer* operation there is no learning automaton in $cell_i$ or $cell_j$.

5. Experimental results

We implemented all algorithms in Java programming language. The *NetBean* was used as IDE to implement the algorithms. All experiments were performed on a Pentium IV 3.0 GHz PC, with 4 GB of RAM, running under Windows 8. Results reported are averages

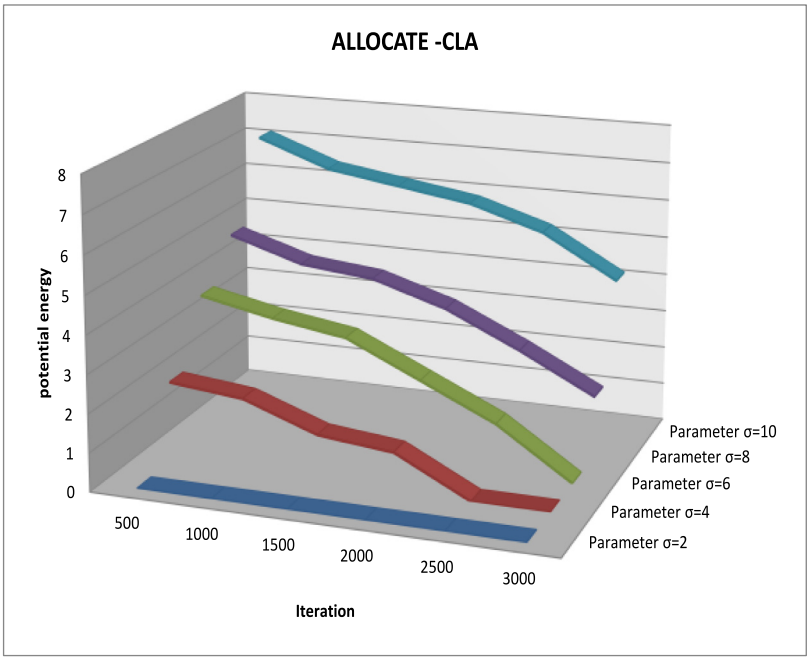

Fig. 40. The impact of σ on *potential energy* of *ALLOCATE-CLA*.

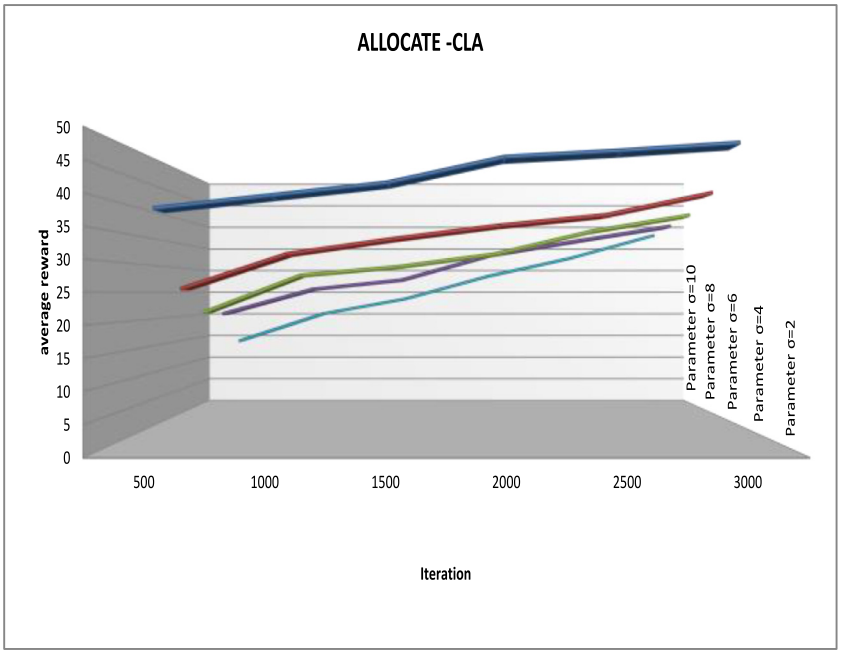

Fig. 41. The impact of σ on *average reward* of *ALLOCATE-CLA*.

over 20 different runs. For the proposed algorithms, each node initially is equipped with a variable structure learning automaton of type L_{RI} with reward parameter $a = 0.001$ and the neighborhood radius of *CLA* is set to 2 and the number of rounds of activation is 200. The proposed algorithm is compared with the algorithms given below.

- *ALLOCATE-CLA-C* algorithm is obtained from *ALLOCATE-CLA* algorithm by replacing the local rule by the following rule as its local rule.

Local rule: The local rule of cell_i sets the reinforcement signal to 1, if the learning automaton of a node_j (returned by the Node_Categorizer function as candidate node) has selected "set the label to hub node", and the other learning automata of cell_i have selected "set the label to spoke node". The local rule of cell_i returns 0, otherwise.

The other parts of the *ALLOCATE-CLA-C* algorithm are similar to *ALLOCATE-CLA* algorithm. *ALLOCATE-CLA-C* is designated to study the performance of the proposed algorithm when the global environment is absent. Note that when the global environment is removed, the *CLA* of the *ALLOCATE-CLA-C* algorithm becomes a closed *CLA*.

- *ALLOCATE-CLA-D* algorithm is a version of *ALLOCATE-CLA* in which *CLA* is replaced with a *pure-chance CLA*. *A* pure-chance *CLA* is a *CLA* in which, each learning automaton is replaced with

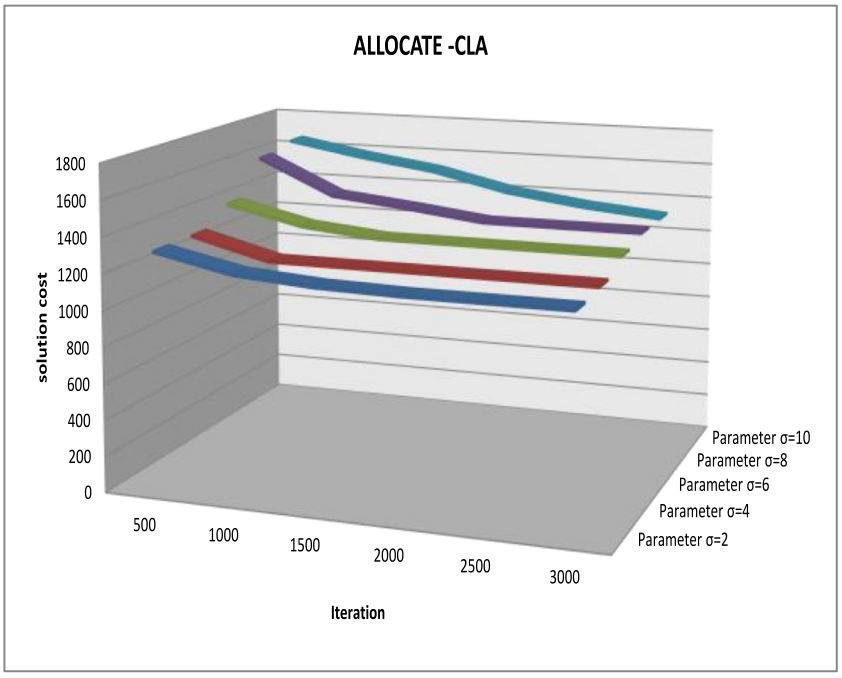

Fig. 42. The impact of σ on *solution cost* of *ALLOCATE-CLA*.

a pure-chance automaton. *ALLOCATE-CLA-D* in which the learning is absent will be used to study the learning capability of CLA on the performance of *ALLOCATE-CLA*.

- An algorithm reported in [27] that we call it *Zarandi's algorithm*. This algorithm is the only reported algorithm for *USA-HLP* when the distances are imprecise.
- A genetic algorithm reported in [32] that we call it *Topcuoglu's algorithm*. This algorithm is a best known algorithm for *USA-HLP* when the distances are precise.

The algorithms are tested on a noisy version of CAB (Civil Aeronautics Board) dataset which we call it imp-CAB (σ). imp-CAB (σ) is obtained from CAB by adding a Gaussian noise with mean 0 and standard deviation σ to d_{ij}, the actual distance between $node_i$ and $node_j$. CAB dataset contains information of airline passenger among 25 cities in 1970 and is a well-known dataset for the allocation hub problem [50–52].

In order to study the performance of the proposed algorithm, four metrics: *entropy, potential energy,* and *solution cost* are used. These metrics are computed using equations Eqs. (8), (7), and (1) respectively. The average reward of the CLA is also analyzed. In Eq. (1), the parameters α_t, α_c, α_d, n, and f_i are set to 0.4, 1, 1, 15, and 100, respectively.

Experiment 1

This experiment is conducted to study the impact of parameter r on the performance of the proposed algorithm *ALLOCATE-CLA* with respect to *entropy, potential energy, average reward,* and *solution cost*. For this purpose, the proposed algorithm is tested for $r = 0.1$, 0.2, 03, 0.4 and 0.5. Figs. 19–22 show the impact of parameter r on performance of *ALLOCATE-CLA* algorithm during its operation when imp-CAB(2) is used as dataset. Figs. 23–26 show the impact of parameter r on performance of *ALLOCATE-CLA* algorithm when the algorithm is terminated at iteration 3000 and imp-CAB(σ) with $\sigma = 2$, 4, 6, 8, and 10 is used as dataset. According to the obtained results, we may conclude the following.

- From Figs. 19 and 20, in which the *entropy* and *potential energy* are shown for different values of parameter r when $\sigma = 2$, indicates that lower value of r results in lower *entropy* and *potential energy* for *ALLOCATE-CLA*.

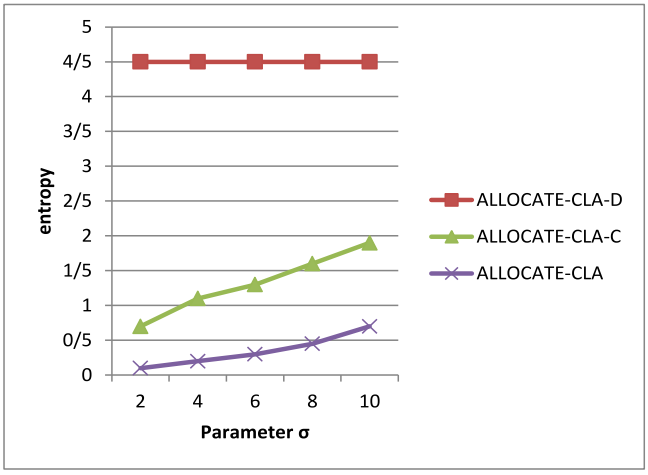

Fig. 43. The impact of σ on *entropy* for different algorithms.

- From Figs. 21 and 22, in which the *average reward* and *solution cost* are shown for different values of r when $\sigma = 2$, indicates that setting r to 0.3 results in the highest *average reward* and *solution cost* for *ALLOCATE-CLA*
- Fig. 23 which plots the value of *entropy* for different values of r and σ at the termination of the algorithm, indicates that for lower value of σ, changes in the actions taken by the learning automata are fewer as compared to higher value of σ. This means that fewer numbers of iterations is required by *ALLOCATE-CLA* in order to find the solution. Also we may say that when σ is high, high value for r must be selected in order for *ALLOCATE-CLA* to find the solution with fewer numbers of iterations.
- Fig. 24 which plots the value of *potential energy* for different values of r and σ at the termination of *ALLOCATE-CLA*, indicates that for lower value of σ, changes in the structure of CLA is fewer as compared to higher value of σ. This means that *ALLOCATE-CLA* when σ is lower requires fewer numbers of iterations in order to reach a fixed structure for CLA. Once

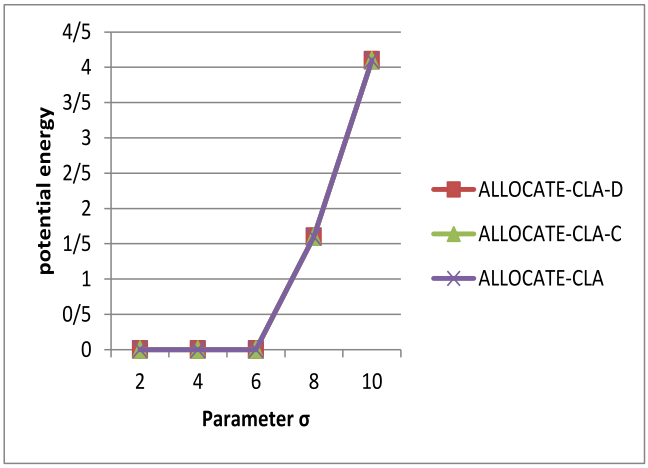

Fig. 44. The impact of σ on *potential energy* for different algorithms.

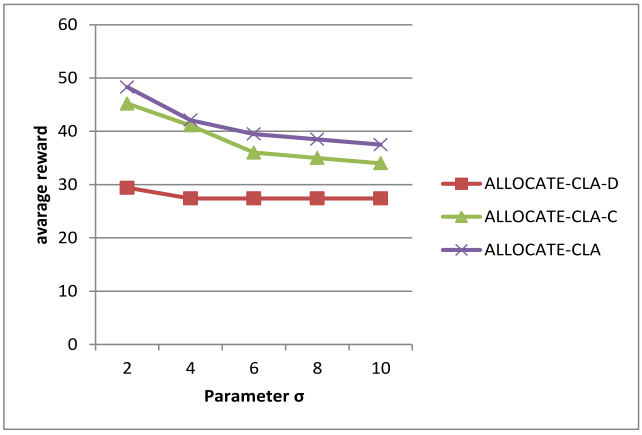

Fig. 45. The impact of σ on *average reward* for different algorithms.

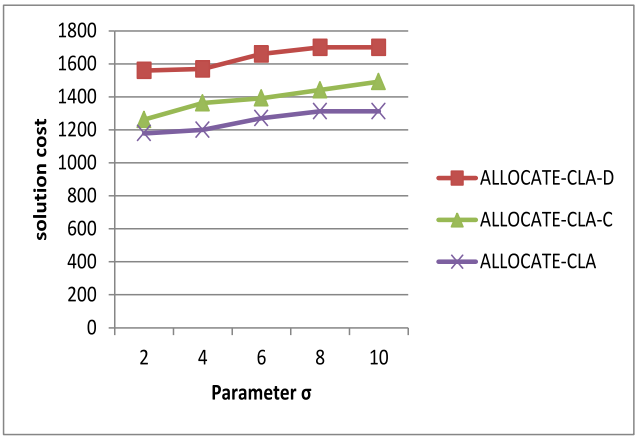

Fig. 46. The impact of σ on *solution cost* for different algorithms.

the structure for CLA becomes fixed the process of labeling the nodes will be done more quickly. Also we may say that when σ is higher, higher value must be selected for r in order for the algorithm to reach a fixed structure for the CLA more quickly.

- From Fig. 25 and 26, we can say that regardless of the value of σ, *solution cost* reaches its lowest value and *average reward* reaches its highest value when $0.2 \leq r \leq 0.3$.

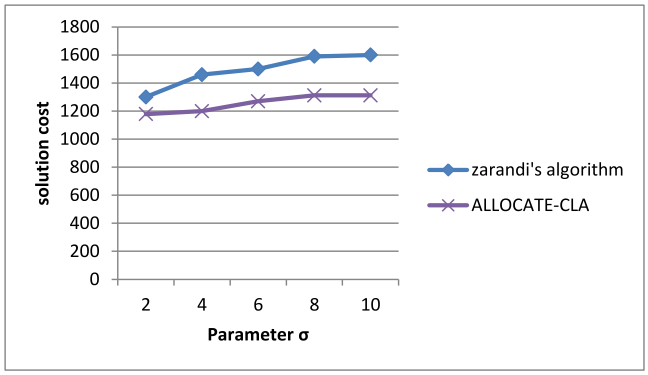

Fig. 47. The impact of σ on solution cost for zarandi's algorithm and *ALLOCATE-CLA*.

- Increasing the value of parameter r decreases the quality of the solution found by *ALLOCATE-CLA* with respect to *average reward* and *solution cost*.

Experiment 2

This experiment is conducted to study the effect of learning capability of learning automata and the global environment on the performance of the proposed algorithm. For this purpose, the proposed algorithm *ALLOCATE-CLA* is compared with *ALLOCATE-CLA-C* and *ALLOCATE-CLA-D* with respect to *entropy, potential energy, average reward,* and *solution cost*. In this experiment, the parameter *r* is set to 0.3 and imp-CAB(2) is used as dataset. The results obtained are reported in Figs. 27–30. According to the results of this experiment, we may conclude the following.

- In terms of *entropy, average reward*, and *solution cost, ALLOCATE-CLA-D* algorithm performs worse than the *ALLOCATE-CLA* algorithm (Figs. 27, 29, and 30). This is because in *ALLOCATE-CLA-D* the learning is absent and hence algorithm is not able to improve its performance as it proceeds.
- In terms of *entropy, average reward*, and *solution cost, ALLOCATE-CLA-C* algorithm performs worse than the *ALLOCATE-CLA* algorithm (Figs. 27, 29, and 30). This is because of absence of the global environment in *ALLOCATE-CLA-C* which provides global information regarding the environment.
- The *entropy* for *ALLOCATE-CLA* and *ALLOCATE-CLA-C* algorithms are high at initial rounds of the simulation, but gradually decrease (Fig. 27). The *entropy* of *ALLOCATE-CLA* is lower than both *ALLOCATE-CLA-C* and *ALLOCATE-CLA-D* algorithms. Lower *entropy* mean lower rates of changes in the actions selected by learning automata residing in the cells of the CLA which indicates lower rates of changes in the labels given to the nodes during the execution of *ALLOCATE-CLA*. The *entropy* of *ALLOCATE-CLA-D* algorithm remains unchanged during its execution. This is because of the fact that the probability vectors of the learning automata of CLA of *ALLOCATE-CLA-D* algorithm remains unchanged as it interacts with the environment.
- The *potential energy* of all algorithms are approaching zero which means that the cellular structure of the *CLA* is approaching to a fixed structure (Fig. 28).
- In terms of *average reward, ALLOCATE-CLA*, and *ALLOCATE-CLA-C* algorithms perform better than *ALLOCATE-CLA-D* algorithm (Fig. 29) which indicates that the CLAs used by *ALLOCATE-CLA*, and *ALLOCATE-CLA-C* algorithms are expedient. In other words, the *CLA* of both algorithms gradually improves their performances and receive more rewards from the environment as they proceed.
- Since the *average reward* received by the *CLA* of *ALLOCATE-CLA* algorithm is higher than the CLA of *ALLOCATE-CLA-C* algo-

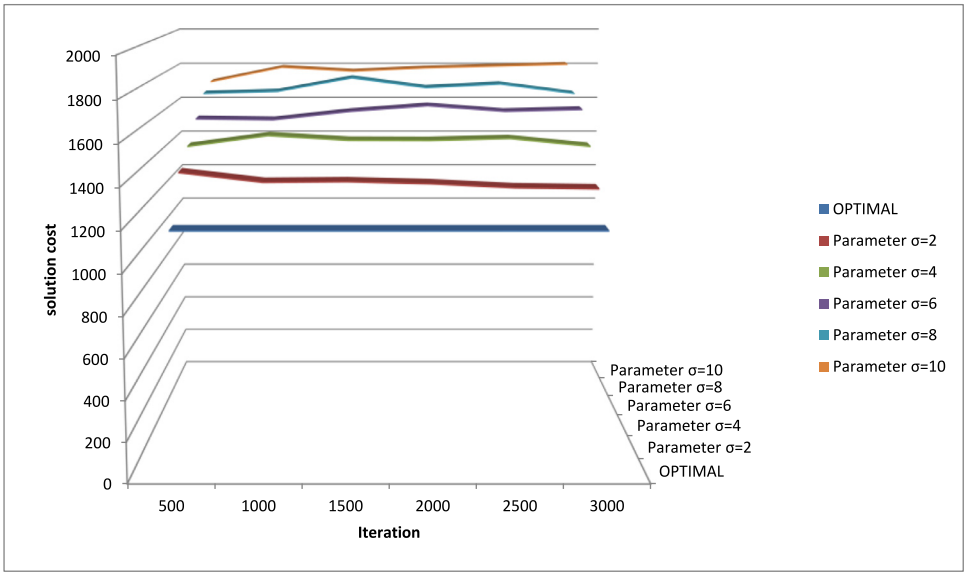

Fig. 48. The impact of parameter σ on solution cost of *Topcuoglu's algorithm*.

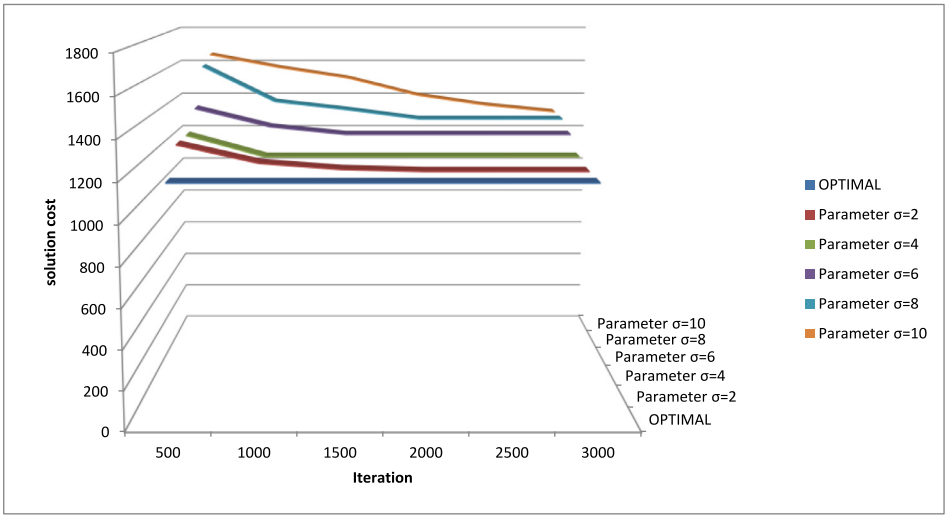

Fig. 49. The impact of parameter σ on solution cost of *ALLOCATE-CLA algorithm*.

rithm, we can say that *ALLOCATE-CLA* is more expedient than *ALLOCATE-CLA-C*.

Experiment 3

This experiment is conducted to study the effect of σ in imp-CAB(σ) on the performance of the *ALLOCATE-CLA, ALLOCATE-CLA-C, ALLOCATE-CLA-D*. For this purpose, we test the algorithms with imp-CAB(σ) for $\sigma = 2$, 4, 6, 8, and 10. Parameter r is set to 0.3. Figs. 31–42 study the performance of *ALLOCATE-CLA, ALLOCATE-CLA-C,* and *ALLOCATE-CLA-D* during their operations with respect *entropy, potential energy, average reward,* and *solution cost and* Figs. 43–46 show the values of *entropy, potential energy, average reward,* and *solution cost* of *ALLOCATE-CLA, ALLOCATE-CLA-C, ALLOCATE-CLA-D* at their terminations (iteration 3000) for different values of parameter σ. According to the results, we may conclude the following.

- In terms of *solution cost, ALLOCATE-CLA* performs better than *ALLOCATE-CLA-C,* and *ALLOCATE-CLA-D* (Fig. 46).
- Increasing the value of σ results in decreasing the *average reward* and increasing the *entropy, potential energy and solution cost* for two algorithms *ALLOCATE-CLA-C,* and *ALLOCATE-CLA* (Figs. 35–46). This means that increasing the amount of noise causes the algorithm to work harder to find the solution.
- For *ALLOCATE-CLA-D* increasing the value of σ results in decreasing the *average reward* and increasing the *solution cost* and *potential energy* (Figs. 32–34).
- *ALLOCATE-CLA* performs better than *ALLOCATE-CLA-D and ALLOCATE-CLA-C* on imprecise distances with respect to all metrics.

Experiment 4

In this experiment, we compare the performance of *ALLOCATE-CLA* with *zarandi's* algorithm (the only reported algorithm for solving *USA-HLP* for imprecise distances) with respect to *solution cost* for different values of σ. Fig. 47 shows the *solution cost* of *zarandi's* algorithm and *ALLOCATE-CLA* at its termination (iteration 3000) for different values of parameter σ. In order to study the statistical significance of the obtained results, *t-test* is used. Table 2 gives the results of two-tailed *t-test* for *solution cost* of *zarandi's* algorithm and *ALLOCATE-CLA* when $\alpha = 0.05$. In this test, it is assumed that the difference between *zarandi's* algorithm and *ALLOCATE-CLA* is statistically significant, if the value of P-Value is smaller than

Table 2
The results of *t*-test for *solution cost* of *zarandi's* algorithm and *ALLOCATE-CLA*.

	ALLOCATE-CLA		Zarandi's algorithm		P-Value
	Mean	Variance	Mean	Variance	
imp-CAB(2)	1179.8695	27.39563658	1298.416	28.14749895	2.58E−42
imp-CAB(4)	1200.914	14.42663	1456.329	22.24009	1.6E−55
imp-CAB(6)	1268.139	42.30176	1497.309	21.8804	2.45E−48
imp-CAB(8)	1311.593	16.19082	1588.255	19.71239	1.42E−59
imp-CAB(10)	1309.149	35.34158	1598.763	21.19127	4.19E−54

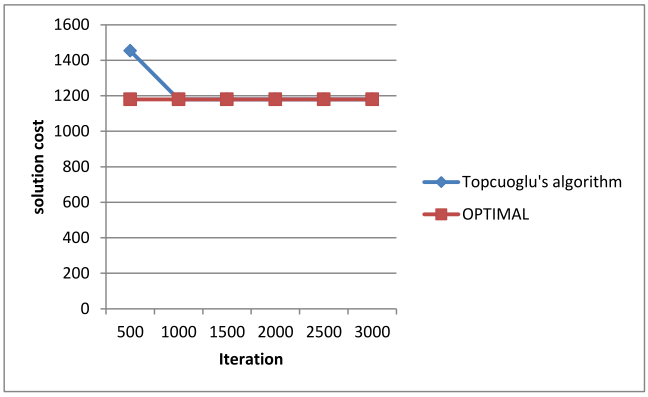

Fig. 50. The solution cost of *Topcuoglu's* algorithm with precise distances ($\sigma = 0$).

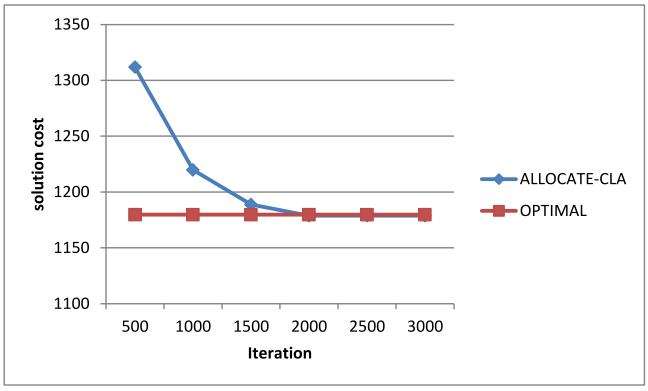

Fig. 51. The solution cost of *ALLOCATE-CLA* algorithm with precise distances ($\sigma = 0$).

α. From the results which are shown in (Table 2 and Fig. 47), we may conclude that in terms of *solution cost, ALLOCATE-CLA* algorithm performs better than *zarandi's* algorithm for all values of σ.

Experiment 5

In this experiment, we test *Topcuoglu's algorithm* [32] the best algorithm for precise distances and *ALLOCATE-CLA* when r = 0.3 for both precise and imprecise distances with respect to the *solution cost*. For this purpose, imp-CAB(0) (CAB data set) is used as precise dataset and imp-CAB(σ) with $\sigma = 2$, 4, 6, 8, and 10 are used as imprecise datasets. The results are given in Fig. 48 and Fig. 49. From the results, we may conclude that for imprecise distances with $\sigma = 2$, and $\sigma = 4$, *ALLOCATE-CLA* converge to a cost near the cost of optimal solution whereas *Topcuoglu's* algorithm for all values of σ cannot find a solution with an acceptable cost. This is because *Topcuoglu's* algorithm has no mechanism for resolving the negative effect of the noise embedded in the distances among nodes. *ALLOCATE-CLA* by using CLA has the ability to remove such an imprecision while trying to find the solution.

Experiment 6

In this exprement, we compare *Topcuoglu's* algorithm with *ALLOCATE-CLA* when the distances are precise. Fig. 50 and Fig. 51 show that both *Topcuoglu's* algorithm and *ALLOCATE-CLA* find the optimal solution. But *ALLOCATE-CLA* requires higher number of iterations as compared to *topcuouglu's* algorithm. Experimentation has also shown that an iteration of *Topcuoglu's* algorithm takes approximately 0.1 ms and an iteration of the proposed algorithm takes approximately 0.3 ms.

6. Conclusion

In this paper, a new open dynamic model of *CLAs* was proposed and then to show the applicability of this model for problem solving, an algorithm for solving single allocation hub location problem with imprecise distances among nodes was designed. The proposed algorithm was compared with *zarandi's* algorithm [27] and *Topcuoglu's* algorithm [32]. The results of simulations show that the proposed algorithm is more robust to imprecise distances as compared to *zarandi's* algorithm which is the only reported algorithm for solving single allocation hub location problem with imprecise distances. One may say that by using *CLA* the algorithm has the ability of removing the imprecision embedded in the distances among nodes while trying to find the solution. In comparison with *Topcuoglu's* algorithm that is the best known algorithm for solving single allocation hub location problem with precise distances, the results of the experiments have shown that the proposed algorithm is able to find the optimal solution when the distances are precise but it takes more time than *Topcuoglu's* algorithm.

As it was previously mentioned, *CLAs* have been used in several applications such as computer networks, image processing, complex networks, optimization, and social networks. The proposed model enables each cell of *CLA* to use more information for better decision making. Therefore, they can be used in many applications. Note that, In order to use this model, the first step is to define a global environment. If the global environment cannot be considered because of distributed or dynamic nature of the application, we may use the closed version of the proposed model.

Note that, the learning automaton is a reinforcement learning model. The reinforcement learning models such as Q-learning [53] and learning automata are used to design self-adaptive systems based on feedback loops. In order to design self-adaptive systems, a general model of feedback loops called MAPE-K is very popular in the literature [54]. Therefore, as future work, two interesting research directions can be perused which are described as follows. In the first direction, we can design a version of the proposed model in which each cell utilizes Q-learning algorithm instead of learning automata. In the second direction, we can generalize the proposed model in which each cell utilizes an algorithm based on MAPE-K loop instead of learning automata.

References

[1] M. Thathachar, P.S. Sastry, Networks of Learning Automata: Techniques for Online Stochastic Optimization, Kluwer Academic Publishers, Dordrecht, Netherlands, 2004.
[2] H. Beigy, M.R. Meybodi, A mathematical framework for cellular learning automata, Adv. Complex Syst. (3) (2004) 295–319.
[3] S. Wolfram, Theory and Applications of Cellular Automata, World Scientific Publication, 1986.

[4] M. Esnaashari, M.R. Meybodi, Dynamic point coverage problem in wireless sensor networks: a cellular learning automata approach, Ad Hoc Sensor Wirel. Netw. 10 (2010) 193–234.
[5] M. Esnaashari, M. Meybodi, A cellular learning automata based clustering algorithm for wireless sensor networks, Sensor Lett. 6 (2008) 723–735.
[6] H. Beigy, M.R. Meybodi, Cellular learning automata with multiple learning automata in each cell and its applications, IEEE Trans. Syst. Man Cybern. Part B 40 (2010) 54–65.
[7] M. Esnaashari, M. Meybodi, A cellular learning automata-based deployment strategy for mobile wireless sensor networks, J. Parallel Distrib. Comput. 71 (2011) 988–1001.
[8] M. Esnaashari, M. Meybodi, Deployment of a mobile wireless sensor network with k-coverage constraint: a cellular learning automata approach, Wireless Netw. 19 (2013) 945–968.
[9] A.M. Saghiri, M.R. Meybodi, A self-adaptive algorithm for topology matching in unstructured peer-to-peer networks, J. Netw. Syst. Manage. (2015), doi:10.1007/s10922-015-9353-9.
[10] A.M. Saghiri, M.R. Meybodi, A distributed adaptive landmark clustering algorithm based on mOverlay and learning automata for topology mismatch problem in unstructured peer-to-peer networks, Int. J. Commun. Syst. (2015), doi:10.1002/dac.2977.
[11] A.M. Saghiri, M.R. Meybodi, An approach for designing cognitive engines in cognitive peer-to-peer networks, J. Netw. Comput. Appl. 70 (2016) 17–40, doi:10.1016/j.jnca.2016.05.012.
[12] Y. Zhao, W. Jiang, S. Li, Y. Ma, G. Su, X. Lin, A cellular learning automata based algorithm for detecting community structure in complex networks, Neurocomputing 151 (2015) 1216–1226.
[13] R. Rastegar, M.R. Meybodi, A new evolutionary computing model based on cellular learning automata, in: IEEE Conference on Cybernetics and Intelligent Systems, Singapore, 2004, pp. 433–438.
[14] R. Rastegar, M.R. Meybodi, A. Hariri, A new fine-grained evolutionary algorithm based on cellular learning automata, Int. J. Hybrid Intell. Syst. 3 (2006) 83–98.
[15] M.H. Mofrad, S. Sadeghi, A. Rezvanian, M.R. Meybodi, Cellular edge detection: Combining cellular automata and cellular learning automata, AEU-Int. J. Electron. Commun. 69 (2015) 1282–1290.
[16] M. Mozafari, M.E. Shiri, H. Beigy, A cooperative learning method based on cellular learning automata and its application in optimization problems, J. Comput. Sci. (2015).
[17] H. Beigy, M.R. Meybodi, Open synchronous cellular learning automata, Adv. Complex Syst. 10 (2007) 527–556.
[18] H. Beigy, M.R. Meybodi, Asynchronous cellular learning automata, Automatica 44 (2008) 1350–1357.
[19] M. Esnaashari, M.R. Meybodi, Irregular cellular learning automata, IEEE Trans. Cybern. 1 (2014), doi:10.1109/TCYB.2014.2356591.
[20] M. Asnaashari, M.R. Meybodi, Irregular cellular learning automata and its application to clustering in sensor networks, in: Proceedings of 15th Conference on Electrical Engineering, IEEE, Tehran, Iran, 2007, pp. 21–28.
[21] M. Vahidipour, M.R. Meybodi, M. Esnaashari, Adaptive petri net based on irregular cellular learning automata and its application in vertex coloring problem systems with unknown parameters, Appl. Intell. (2016), doi:10.1007/s10489-016-0831-x.
[22] M.M. Daliri Khomami, A.R. Rezvanian, M.R. Meybodi, Irregular cellular automata for multiple diffusion, in: Proceedings of the 22th Iranian Conference on Electrical Engineering, Shahid Beheshti University, Tehran, Iran, 2014.
[23] A.M. Saghiri, M.R. Meybodi, On expediency of closed asynchronous dynamic cellular learning automata, J. Comput. Sci. (2017).
[24] M.E. O'Kelly, A clustering approach to the planar hub location problem, Ann. Oper. Res. 40 (1992) 339–353.
[25] R. Zanjirani Farahani, M. Hekmatfar, A. Boloori Arabani, E. Nikbakhsh, Hub location problems: A review of models, classification, solution techniques, and applications, Comput.Ind. Eng. 64 (2013) 1096–1109.
[26] T.-H. Yang, A two-stage stochastic model for airline network design with uncertain demand, Transportmetrica 6 (2010) 187–213.
[27] S. Davari, F. Zarandi, B. Turksen, The incomplete hub-covering location problem considering imprecise location of demands, Scientia Iranica 20 (2013) 983–991.
[28] Y. An, Y. Zhang, B. Zeng, The reliable hub-and-spoke design problem: models and algorithms, Transp. Res. Part B 77 (2015) 103–122.
[29] K. Smith, M Krishnamoorthy, M Palaniswami, Neuralversus traditionalapproaches to the location of interacting Hub facilities, Location Science 4 (1996) 155–171.
[30] J. Klincewicz, Heuristics for the p-Hub location problem, Eur. J. Oper. Res. (1991) 25–37.
[31] A.-H. Sue, A hybrid heuristic for the uncapacitated Hub location problem, Eur. J. Oper. Res. (1998) 489–499.
[32] H. Topcuoglu, F. Corut, M. Ermis, G. Yilmaz, Solving the uncapacitated hub location problem using genetic algorithms, Comput. Oper. Res. (2005) 967–984.
[33] M. Roberto Silva, C.B. Cunha, New simple and efficient heuristics for the uncapacitated single allocation hub location problem, Comput. Oper. Res. 36 (2009) 3152–3165.
[34] J.-F. Chen, A hybrid heuristic for the uncapacitated single allocation hub location problem, Omega 35 (2007) 211–220.
[35] I. Contreras, J.-F. Cordeau, G. Laporte, Stochastic uncapacitated hub location, Eur. J. Oper. Res. 212 (2011) 518–528.
[36] A. Eydi, A. Mirakhorli, An extended model for the uncapacitated single allocation hub covering problem in a fuzzy environment, Proceeding of the International Multi Conference of Engineers and Computer Scientists, 2012.
[37] K.S. Narendra, M.A.L. Thathachar, Learning Automata: An Introduction, Prentice-Hall, Englewood Cliffs, NJ, 1989.
[38] Y. Guo, G. Hao, L. Shenghong, A set of novel continuous action-set reinforcement learning automata models to optimize continuous functions, Appl. Intell. 46 (2017) 845–864.
[39] M.H. Mofrad, O. Jalilian, A.R. Rezvanian, M.R. Meybodi, Service level agreement based adaptive grid superschedulin, Future Gener. Comput. Syst. 55 (2016) 62–73.
[40] M. Ghavipour, M.R. Meybodi, Trust propagation algorithm based on learning automata for inferring local trust in online social networks, Knowl. Based Syst. (2017).
[41] A.R. Rezvanian, M.R. Meybodi, Sampling algorithms for stochastic graphs: a learning automata approach, Knowl. Based Syst. (2017), doi:10.1016/j.knosys.2017.04.012.
[42] A. Rezvanian, M.R. Meybodi, Stochastic graph as a model for social networks, Comput. Hum. Behav. 64 (2016) 621–640.
[43] M. Vahidipour, M.R. Meybodi, Cellular adaptive Petri net based on learning automata and its application to the vertex coloring problem, Discrete Event Dyn. Syst. (2017) 1–32.
[44] B. Damerchilu, M.S. Norouzzadeh, M.R. Meybodi, Motion estimation using learning automata, Mach. Vis. Appl. 27 (2016) 1047–1061.
[45] M. Mahdaviani, J.K. Kordestani, A. Rezvanian, M.R. Meybodi, LADE: learning automata based differential evolution, Int. J. Artif. Intell. Tools 24 (2015) 1550023.
[46] M.M.D. Khomami, A.R. Rezvanian, N. Bagherpour, M.R. Meybodi, Minimum positive influence dominating set and its application in influence maximization: a learning automata approach, Appl. Intell. (2017) 1–24.
[47] M.M.D. Khomami, A.R. Rezvanian, M.R. Meybodi, Distributed learning automata-based algorithm for community detection in complex networks, Int. J. Mod. Phys. B 30 (2016) 1650042.
[48] D.E. Knuth, The Art of Computer Programming, Addison-Wesley, 1977.
[49] F. Aurenhammer, Voronoi diagrams—a survey of a fundamental geometric data structure, ACM Comput. Surv. (CSUR) 23 (1991) 345–405.
[50] M.E. O'kelly, A quadratic integer program for the location of interacting hub facilities, Eur. J. Oper. Res. 32 (1987) 393–404.
[51] A.T. Ernst, M. Krishnamoorthy, Effcient algorithms for the uncapacitated single allocation p-Hub median problem, Loc. Sci. 4 (1996) 139–154.
[52] J. Beasley, OR-library: distributing test problems by electronic mail, J. Oper. Res. 41 (1990) 1069–1072.
[53] R.S. Sutton, A.G. Barto, Reinforcement learning: An introduction, Univ Press, Cambridge, 1998.
[54] D.G.D.L. Iglesia, D. Weyns, MAPE-K formal templates to rigorously design behaviors for self-adaptive systems, ACM Trans. Auton. Adap. Syst. 10 (2015) 15.