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

Ali Mohammad Saghiiri*, Mohammad Reza Meybodi

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

A R T I C L E   I N F O

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

A B S T R A C T

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].
• 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).

\[
\begin{align*}
\min \, f(y) &= \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} W_{ij} \lambda_{ik} y_{ik} \\
&+ \sum_{i=1}^{n} \sum_{j=1}^{n} W_{ij} \lambda_{ik} y_{jk} \\
&+ \sum_{k=1}^{n} \sum_{l=1}^{n} y_{ik} \lambda_{kl} y_{jk} W_{ij} \\
&+ \sum_{j=1}^{n} y_{ij} e_j
\end{align*}
\]

subject to

\[
\begin{align*}
\sum_{k=1}^{n} y_{ik} &= 1 \quad \forall i \\
y_{ik} - y_{jk} &\geq 0 \quad \forall i, k \\
y_{ik} &\in \{0, 1\} \quad \forall i, k
\end{align*}
\]
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 some of which are described as below.

- 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 imprecision issue.

**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.
      o The cell set its attribute.
      o The cell and its neighboring cells compute their restructuring signals using the local rule (Fi).
    - Structure updating phase: In this phase, the cell performs the following steps.
      o The neighborhood structure of the cell is updated using the structure updating rule (Fi) if the value of the restructuring signal of that cell is 1.
    - State updating phase: In this phase, the cell performs the following steps.
      o 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.
      o The local rule (Fi) is applied and a reinforcement signal is generated.
      o 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.

```
Algorithm Activate()
Input
   celli
Notations
   F_i denotes the local rule
   F_j denotes the structure updating rule
   \( \psi_i \) denotes the attribute of celli
   \( \phi_i \) denotes the state of celli
   \( \zeta_i \) denotes the restructuring signal of celli
   \( N_i \) denotes the set of neighbors of celli
   \( \beta_i \) denotes the reinforcement signal of the learning automata of celli.

Begin
   \% preparation phase\%
   Set the attribute of the celli, \( \psi_i \) using \( F_i \);
   Ask from neighboring cells of celli to compute their restructuring signals;
   Gather the restructuring signals of the neighboring cells;
   \% structure updating phase\%
   If \( (\zeta_i = 1) \) Then
      Compute \( N_i \) and \( \psi_i \) using \( F_i \);
   EndIf
   \% state updating phase\%
   Each learning automaton of celli chooses one of its actions;
   Set \( \phi_i \); set \( \phi_i \) to be the set of actions chosen by the set of learning automata in celli;
   Compute \( \beta_i \) using \( F_i \);
   Update the action probabilities of learning automata of celli using \( \beta_i \);
End
```

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

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 imprecision issue.
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.

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, and its neighbors

**Output**
Immediate neighbors and attribute of cell,

**Notations**
Let group be the set nodes of cell,
Let group be the set nodes of cell,
Let d-nodes be the set of departed nodes of cell,
Let c-nodes be the candidate node of cell,

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

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

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 (β, α, P, G), where β a set of inputs actions (called response or reinforcement signal), α 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.

\[
p_i(k+1) = p_i(k) + a(1 - p_i(k))
\]

(5)

\[
p_j(k+1) = p_j(k) - a p_j(k), \quad \forall j \neq i
\]

(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 (β = 1), and (6) for an unfavorable response (β = 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.
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 εP); 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
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:

\[ \text{OADCLA} - VL = (G, A, \Psi, \Phi, N, E, F_1, F_2) \]

where:

- \( G = (V, E) \) is an undirected graph which determines the structure of OADCLA-VL where \( V = \{ \text{cell}_1, \text{cell}_2, \ldots, \text{cell}_n \} \) is the set of vertices and \( E \) is the set of edges. \( A = \{ LA_1, LA_2, \ldots, LA_n \} \) is a set of LAs. A subset of set \( A \) is assigned to a cell.
- \( N = \{ N_1, N_2, \ldots, N_n \} \) where \( N_i = \{ \text{cell}_j \in V | \text{dist}(\text{cell}_i, \text{cell}_j) < \theta_i \} \) where \( \theta_i \) is the neighborhood radius of \( \text{cell}_i \) and \( \text{dist}(\text{cell}_i, \text{cell}_j) \) is the length of the shortest path between \( \text{cell}_i \) and \( \text{cell}_j \) in \( G \). \( N_i^1 \) determines the immediate neighbors of \( \text{cell}_i \).
- \( \Psi = \{ \Psi_1, \Psi_2, \ldots, \Psi_n \} \). \( \Psi_i = \{ (j, X_j, C_j) | \text{cell}_j \in N_i \} \) denotes the attribute of \( \text{cell}_i \) where \( X_j = \{ x_1, x_2, \ldots, x_s \} \) and \( C_j \subseteq A \). \( \{ x_1, x_2, \ldots, x_s \} \) is the set of allowable attributes. \( \Psi_i^1 \) determines the attribute of \( \text{cell}_i \) when \( \theta_i = 1 \). \( \Psi_i^l \) determines the attribute of \( \text{cell}_i \) when \( \theta_i = l \).
- \( \Phi = \{ \Phi_1, \Phi_2, \ldots, \Phi_n \} \) where \( \Phi_i = \{ (j, k, \alpha) | \text{cell}_j \in N_i \text{ and action } \alpha_i \text{ has been chosen by } LA_k \in C_j \} \) denotes the state of attributes.
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, $\Phi^i_l$ determines the state of cell $i$ when $\theta_i = 1$. $\Phi^i_l$ determines the state of cell $i$ when $\theta_i = 1$.

- $E^C$ 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 $(N, \Phi, \Psi)$ and its output is $Q(E^C)$.

- $F_1: (\Phi, \Psi, Q(E^C)) \rightarrow (\beta, \zeta)$ 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 $< \Phi^i_l, \Psi^i_l, Q(E^C) >$ and returns $< \beta^i_l, \zeta^i_l >$. The reinforcement signal is used to update the LA of that cell. Note that, $\zeta^i_l$ is the restructuring signal of cell $i$.

- $F_2: (N, \Psi, \zeta) \rightarrow (N^1, \Psi^1)$ is the structure updating rule. In each cell, the structure updating rule finds the immediate neighbors and attributes of that cell. For example, in cell $i$, structure updating rule takes $< N^i_l, \Psi^i_l, \zeta^i_l >$ and returns $N^1_i$ and $\Psi^1_i$.

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.
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

\[ T(t) = \sum_{i=1}^{n} \zeta_i \]  \hspace{1cm} (7)

where \( \zeta_i(t) \) 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-
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

$$H(t) = -\sum_{i=1}^{n} H_i(t)$$

(8)

where $n$ is the number of cells of the CLA, $H_i(t)$ is the entropy of cell $c_{il}$ of CLA. The entropy of cell $c_{il}$ is defined by Eq. (9) given below

$$H_i(t) = \sum_{i\in C_l(t)} H_i(t)$$

(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

$$H_i(t) = \sum_{j=1}^{r_i} p_{ij}(t) \ln(p_{ij}(t))$$

(10)

where $r_i$ is the number of actions of learning automaton $LA_i$, $p_{ij}(t)$ is the probability of selecting action $\alpha_j$ 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].
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 result, 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<sub>j</sub>, group<sub>k</sub>, group<sub>l</sub>, and group<sub>m</sub> are the groups of cell<sub>j</sub>, cell<sub>k</sub>, cell<sub>l</sub>, and cell<sub>m</sub>, respectively and node<sub>0</sub>, node<sub>1</sub>, node<sub>2</sub>, and node<sub>3</sub> determined by the algorithm to be the hub nodes of group<sub>j</sub>, group<sub>k</sub>, group<sub>l</sub>, and group<sub>m</sub>, 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. 29. Comparison of ALLOCATE-CLA, ALLOCATE-CLA-C and ALLOCATE-CLA-D with respect to average reward.](image)

![Fig. 30. Comparison of ALLOCATE-CLA, ALLOCATE-CLA-C and ALLOCATE-CLA-D with respect to solution cost.](image)

![Fig. 31. The impact of \( \sigma \) on entropy of ALLOCATE-CLA-D.](image)
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 \textit{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 \textit{distance evaluator} to resolve the measurement error of distance evaluation among nodes. In each group, a team of \textit{LAs} are organized for labeling the nodes.

To complete the description of the algorithm we need to describe the \textbf{Global environment}, \textbf{Local rule}, and \textbf{structure updating rule} for the CLA used by function \textit{Activate(cell)} called by Algorithm \textit{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
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 cell\(i\) as input and then returns reinforcement signal \(\beta_i\) and restructuring signal \(\zeta_i\) of cell\(i\).
  - In cell\(i\), the restructuring signal \(\zeta_i\) is set to 1 if there is at least one departed node (returned by function Node_Categorizer (Fig. 12) and returns 0 otherwise.
  - In cell\(i\), the reinforcement signal \(\beta_i\) is computed by an algorithm described as follows. The reinforcement signal is 1, if the LA of node\(j\) (returned by the Node_Categorizer function as c-node) has selected “set the label to hub node”, other LAs of cell\(i\) 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-
distance_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\(_j\). cell\(_j\) is a neighboring cell of cell\(_i\) whose hub node has the least distance to the hub node of cell\(_i\).
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_k is returned by Node_Categorizer as a departed node for group_j. In group_i, if node_k 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_k 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_w, cell_j, cell_l, 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.
over 20 different runs. For the proposed algorithms, each node initially is equipped with a variable structure learning automaton of type $L_{w}$ 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, 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, have selected “set the label to spoke node”. The local rule of cell, 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
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, average reward, 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 \( \alpha_i, \alpha_c, \alpha_d, n, \) 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 \( r=0.1, 0.2, 0.3, 0.4 \) 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.

- 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 \( \sigma \) at the termination of the algorithm, indicates that for lower value of \( \sigma \), changes in the actions taken by the learning automata are fewer as compared to higher value of \( \sigma \). This means that fewer numbers of iterations is required by ALLOCATE-CLA in order to find the solution. Also we may say that when \( \sigma \) 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 \( \sigma \) at the termination of ALLOCATE-CLA, indicates that for lower value of \( \sigma \), changes in the structure of CLA is fewer as compared to higher value of \( \sigma \). This means that ALLOCATE-CLA when \( \sigma \) is lower requires fewer numbers of iterations in order to reach a fixed structure for CLA. Once
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. 44.** The impact of $\sigma$ on potential energy for different algorithms.

**Fig. 45.** The impact of $\sigma$ on average reward for different algorithms.

**Fig. 46.** The impact of $\sigma$ on solution cost for different algorithms.

**Fig. 47.** The impact of $\sigma$ on solution cost for zarandi’s algorithm and ALLOCATE-CLA.
Algorithm, we can say that ALLOCATE-CLA is more expedient than ALLOCATE-CLA-C.

**Experiment 3**

This experiment is conducted to study the effect of $\sigma$ in imp-CAB($\sigma$) on the performance of the ALLOCATE-CLA, ALLOCATE-CLA-C, ALLOCATE-CLA-D. For this purpose, we test the algorithms with imp-CAB($\sigma$) 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 to 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 $\sigma$. 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 $\sigma$ 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 $\sigma$ 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 $\sigma$. Fig. 47 shows the solution cost of zarandi’s algorithm and ALLOCATE-CLA at its termination (iteration 3000) for different values of parameter $\sigma$. 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
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 σ = 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 σ = 2, and σ = 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 experiment, 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 topcuoglu’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 pursued 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**


