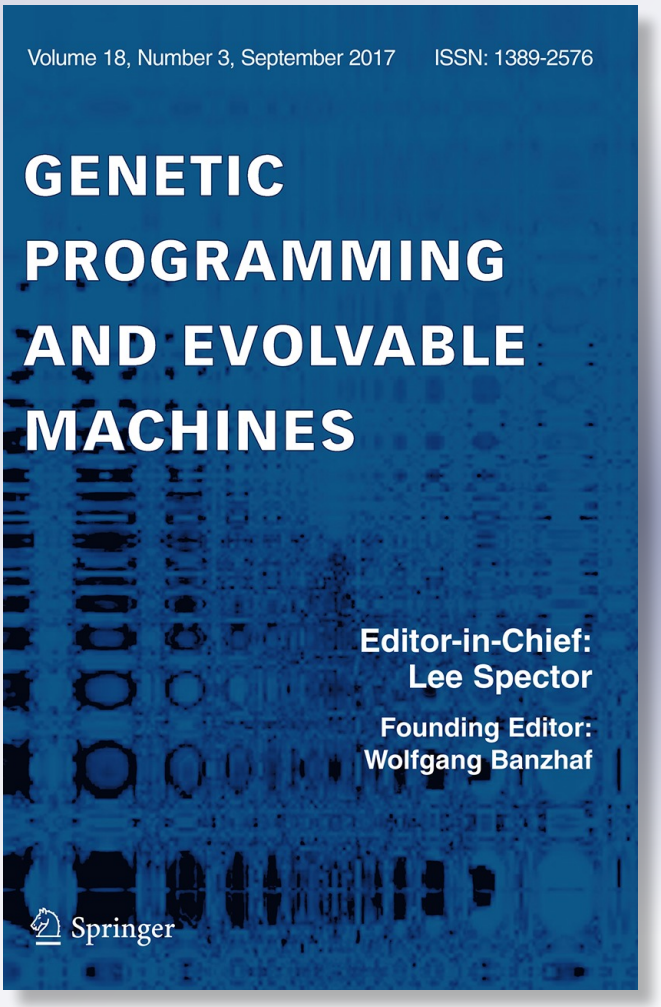
A closed asynchronous dynamic model of cellular learning automata and its application to peer-to-peer networks

Ali Mohammad Saghiri & Mohammad Reza Meybodi

Genetic Programming and Evolvable Machines

ISSN 1389-2576
Volume 18
Number 3

Genet Program Evolvable Mach (2017)
18:313-349
DOI 10.1007/s10710-017-9299-7



Springer

Genet Program Evolvable Mach (2017) 18:313–349
DOI 10.1007/s10710-017-9299-7

A closed asynchronous dynamic model of cellular learning automata and its application to peer-to-peer networks

Ali Mohammad Saghiri[1] · **Mohammad Reza Meybodi**[1]

Received: 13 November 2016 / Revised: 2 February 2017 / Published online: 20 February 2017

Abstract *Cellular Learning Automata* (*CLAs*) are hybrid models obtained from combination of *Cellular Automata* (*CAs*) and *Learning Automata* (*LAs*). These models can be either open or closed. In closed *CLAs*, the states of neighboring cells of each cell called local environment affect on the action selection process of the *LA* of that cell whereas in open *CLAs*, each cell, in addition to its local environment has an exclusive environment which is observed by the cell only and the global environment which can be observed by all the cells in *CLA*. In dynamic models of *CLAs*, one of their aspects such as structure, local rule or neighborhood radius may change during the evolution of the *CLA*. *CLAs* can also be classified as synchronous *CLAs* or asynchronous *CLAs*. In a synchronous *CLA*, all *LAs* in different cells are activated synchronously whereas in an asynchronous *CLA*, the *LAs* in different cells are activated asynchronously. In this paper, a new closed asynchronous dynamic model of *CLA* whose structure and the number of *LAs* in each cell may vary with time has been introduced. To show the potential of the proposed model, a landmark clustering algorithm for solving topology mismatch problem in unstructured peer-to-peer networks has been proposed. To evaluate the proposed algorithm, computer simulations have been conducted and then the results are compared with the results obtained for two existing algorithms for solving topology mismatch problem. It has been shown that the proposed algorithm is superior to the existing algorithms with respect to communication delay and average round-trip time between peers within clusters.

✉ Mohammad Reza Meybodi
mmeybodi@aut.ac.ir

Ali Mohammad Saghiri
a_m_saghiri@aut.ac.ir

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

Keywords Cellular learning automata · Dynamic cellular learning automata · Peer-to-peer networks · Landmark clustering algorithm

1 Introduction

Cellular Automata (*CAs*) are discrete dynamical models which are composed of a large number of independent and identical cells. In these models, the cells are arranged into a lattice. Each cell selects a state from a set of states. In a cell, the previous states of a set of cells, including that cell itself, and their neighbors determines the new state of that cell [1]. In [2], it was shown that a simple one-dimensional *CA* has the same computational power as a Turing machine. These models have found applications in some areas such as peer-to-peer networks [3], and social modeling [4]. On the other hand, *Learning Automata* (*LAs*) can be used as models for adaptive online decision making in unknown environments. A *LA* has a finite set of actions. Its goal is to learn the optimal action through repeated interaction with the environment. An optimal action is an action with the highest probability of getting reward from the environment. *LAs* have found applications in several areas such as sensor networks [5], channel assignment [6], and peer-to-peer networks [7–11], to mention a few.

Cellular Learning Automata (*CLAs*) are hybrid models based on *CAs* and *LAs* [12]. In a *CLA*, each cell of the *CA* is equipped with a *LA*. In a cell, the action selected by the *LA* of the cell determines the state of the cell. In the *CLA*, a local rule is defined to determine the response of the *LA* of each cell. Each cell utilizes the local rule to generate the response of its *LA* according to the state of itself and the states of its neighbors. Since, *CLAs* inherit the distributed computation from the *CAs* and learning in unknown environments from the *LAs*, they can be useful in many problems in distributed systems such as computer networks [12–15]. Note that, *CLAs* have been also used in applications such as social networks [16], evolutionary computing [17, 18], and optimization problems [19].

Several models of *CLAs* are reported in the literature. The reported models can be classified into two main classes as described below.

- Static *CLAs* (*SCLAs*): In *SCLAs*, the structure of the cells remains fixed during the evolution of the *SCLA* [16, 19–23]. *SCLAs* can be either closed or open. In closed *CLAs*, the states of neighboring cells of each cell called local environment affect on the action selection process of the *LA* of that cell whereas in open *CLAs*, each cell, in addition to its local environment has an exclusive environment which is observed by the cell only and the global environment which can be observed by all the cells in the *CLA*. *SCLAs* can be further classified as either synchronous or asynchronous. In a synchronous *SCLA*, all the cells update their states at the same time [20]. 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 update their states and the states of the rest of the cells remain unchanged [21]. In [22], a model of *SCLA* with multiple learning automata in each cell was reported. In this model,

the set of *LAs* of a cell remains fixed during the evolution of the *SCLA*. *SCLAs* depending on its structure can also be classified as regular [12] or irregular [23]. In irregular *SCLAs*, the structure regularity assumption is removed.

- Dynamic *CLAs* (*DCLAs*): In a *DCLA*, one of its aspects such as structure, local rule or neighborhood radius may change over time. *DCLAs* can be classified as either closed [14, 15, 24] or open. *DCLAs* can also be classified as synchronous or asynchronous. In synchronous *DCLAs*, all *LAs* in different cells are activated synchronously whereas in asynchronous *DCLA*s, the *LAs* in different cells are activated asynchronously. All the reported *DCLAs* are closed and asynchronous [14, 15, 24].

Since the goal of this paper is to propose a *CLA* based landmark clustering algorithm for solving the topology mismatch problem in peer-to-peer networks, the rest of the introduction section is devoted to more detailed discussion on the topology mismatch problem.

Peer-to-peer networks construct overlay networks over underlay networks, and can be structured or unstructured. In these networks, each peer directly communicates with other peers, and continually joins and leaves the network. Designing topology management algorithms for these networks is a challenging problem because the network dynamicity caused by joining and leaving peers affects the performance of the management algorithm. Topology mismatch problem is an important problem in unstructured peer-to-peer networks [25–30]. In these networks, the peers choose their neighbors without considering the underlay positions. Therefore the topology of overlay network may have mismatches with its underlying network. The management algorithms for solving the topology mismatch problem can be classified into three classes [11]: the algorithms in which each peer in the network uses some services such as *GPS* to gather information about the locations of peers, Landmark clustering algorithms, and the algorithms in which each peer in the network uses local information about its neighbors. Since in this paper, we propose a new landmark clustering algorithm, the landmark clustering algorithms are described in more details in the next paragraph.

In Landmark clustering algorithms such as those reported in [25, 27, 31–37], the overlay network is organized by clusters in which each cluster has one landmark peer and several ordinary peers. In the landmark clustering algorithms, the peers organize themselves into clusters such that peers that get in within a given cluster are relatively closed to each other in terms of communication delay. These algorithms can be classified into two subclasses: algorithms that use static landmark selection methods [27, 35–37] and algorithms that use dynamic landmark selection methods [7, 25, 34]. The dynamic landmark selection methods can be further classified into two subclasses: algorithms that use non-adaptive landmark selection methods such as *mOverlay* reported in [25], and algorithms that use adaptive landmark selection methods such as *lOverlay* reported in [7]. A drawback of *lOverlay* is that it has many parameters that must be tuned.

In this paper, a new closed asynchronous dynamic model of *CLA* whose structure and the number of *LAs* of each cell may vary with time is suggested and then is used to design an adaptive landmark clustering algorithm for solving the topology

mismatch problem in unstructured peer-to-peer networks. In this model, in contrast to the existing models of *CLA*, the number of *LAs* in each cell may vary with time. To show the superiority of the proposed landmark clustering algorithm, it is compared with *mOverlay* [25] and *lOverlay* [7] algorithms. Both the proposed algorithm and the *lOverlay* algorithm are adaptive landmark selection methods. The *lOverlay* algorithm uses a hybrid mechanism based on *mOverlay* and learning automata whereas the proposed algorithm uses a mechanism based on *CLA* and *Voronoi* diagrams. In the literature, *lOverlay* is the only algorithm which uses adaptive landmark selection method. One of the advantages of the proposed algorithm is that it has fewer parameters than *lOverlay* algorithm. Simulation results have shown that the proposed algorithm performs better than the existing algorithms with respect to communication delay and average round-trip time between peers within clusters.

The rest of this paper is organized as follows. In Sect. 2, we briefly introduce learning automata, landmark clustering algorithms, and *Voronoi* diagrams. In Sect. 3 a new model of dynamic cellular learning automata is introduced and in Sect. 4, an adaptive landmark clustering algorithm based on this model is described for solving topology mismatch problem in unstructured peer-to-peer networks. Section 5 reports the results of experiments, and Sect. 6 concludes the paper.

2 Preliminaries

In this section, in order to provide basic information for the remainder of the paper, we present a brief overview of learning automata, landmark clustering algorithms, and *Voronoi* diagrams.

2.1 Learning automata

Learning Automata (*LAs*) refer to models for adaptive online decision making in unknown environments. The relationship between a *LA* and its environment is shown in Fig. 1. A *LA* has a finite set of actions. Its goal is to learn to choose the optimal action through repeated interaction with the environment utilizing a learning process. The learning process of the *LA* is described as follows. The *LA* randomly chooses an action from its action set and 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

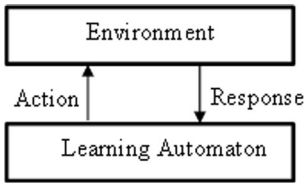


Fig. 1 Learning automaton (*LA*)

and then the learning procedure 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.

LAs can be classified into two classes, fixed and variable structure *LAs* [38, 39]. Variable structure *LAs* which is used in this paper is represented by sextuple $\langle \underline{\beta}, \underline{\phi}, \underline{\alpha}, P, G, T \rangle$, where $\underline{\beta}$ a set of input actions (called response or reinforcement signal), $\underline{\phi}$ is a set of internal states, $\underline{\alpha}$ a set of outputs, $P = \langle p_1, p_2, \ldots, p_r \rangle$ denotes the state probability vector, G is the output mapping, and T is learning algorithm. The learning algorithm is used to modify the probability vector.

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

$$\begin{aligned} p_i(k+1) &= (1-b) \cdot p_i(k) \\ p_j(k+1) &= \frac{b}{r-1} + (1-b) \cdot p_j(k), \quad \forall j \neq i \end{aligned} \tag{2}$$

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 (1) for a favorable response ($\beta = 1$), and (2) for an unfavorable response ($\beta = 0$). Two 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 learning automaton. 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.

2.2 Landmark clustering algorithms

Consider n peers $peer_1, peer_2, \ldots, peer_n$ which are connected to each other through an overlay network over an underlay network. In the landmark clustering algorithms, the overlay network is organized by clusters. Graph $G = (V, E)$ which $V = \{peer_1, peer_2, \ldots, peer_n\}$ is a set of peers and $E \subseteq V \times V$ is a set of links connecting the peers is used to represent the topology of the overlay network. This graph is called overlay graph. Each peer in the overlay network is mapped to a position in the underlay network according to a one-to-one function. In the underlay network, The set of positions is denoted by $V^u = \{pos_1, pos_2, , \ldots, pos_n\}$. Graph $G^u = (V^u, E^u)$ in which $E^u \subseteq V^u \times V^u$ is a set of links connecting the positions is used to represent the topology of the underlay network. This graph is called underlay graph. In the landmark clustering algorithm, the peers are divided into several clusters which each cluster has a landmark peer. In this algorithm, each peer plays one of two roles: *Ordinary* and *Landmark*. Each ordinary peer communicates with other ordinary peers using landmark peers. The set of clusters is denoted by $V^c = \{cluster_1, cluster_2, \ldots, cluster_p\}$. Graph $G^c = (V^c, E^c)$ which $E^c \subseteq V^c \times V^c$ is

a set of links connecting the landmark peer of clusters that is used to represent the topology of the network of the landmark peers. Note that, each cluster in V^c is mapped to several peers in V according to a one-to-many function. The goal of landmark clustering algorithms is to select some peers as landmark peers and reconfigure the links among peers in order to improve the performance of the overlay network in terms of communication delay. The problem of finding a set of landmark peers to minimize the total communication delay is similar to the problem of selecting super-peers in a super-peer network that is proven to be an NP-hard problem [36, 40]. The landmark clustering algorithms try to find an overlay network such that (3) is minimized.

$$\min z = \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} \sum_{k=1}^{n} \sum_{m=1}^{n} \left(d_{ik} + d_{km} + d_{mj}\right).y_{ik}.y_{jm} \tag{3}$$

$$\begin{aligned} &s.t \\ &y_{ij} \leq y_{jj} \quad i,j = 1,\ldots,n \end{aligned} \tag{4}$$

$$\sum_{j=1}^{n} y_{ij} = 1 \quad i,j = 1,\ldots,n \tag{5}$$

$$y_{ij} \in \{0,1\} \quad i,j = 1,\ldots,n \tag{6}$$

In (3), z is the total all-pairs end-to-end communication delay of the overlay network in which d_{ik} is the end-to-end delay from $peer_i$ and $peer_k$ in the underlay network. In (6), y_{ij} indicates whether there is a connection between $peer_i$ and $peer_j$ or not. If there is a connection between $peer_i$ and $peer_j$, the value of variable y_{ij} is equal to 1. If $peer_j$ is chosen as a landmark peer, the value of variable y_{jj} is equal to 1. Constraint (4) indicates that an ordinary peer cannot connect to another ordinary peer. Constraint (5) indicates that each ordinary peer connects to exactly one landmark peer. Since no global knowledge about the network exists and the conditions of the network are highly dynamic, minimizing (3) subject to (4) (5) (6) by the landmark clustering algorithm leading to a challenging problem.

2.3 Voronoi diagrams

In the field of computational geometry, constructing the *Voronoi* diagram for a set of n points in the Euclidean plane is one of the well-known problems. In a Euclidean plane, the *Voronoi* diagram of a set of points is a collection of cells that divide up the plane. Figure 2 shows an example for a *Voronoi* diagram for six points in a plane. Each cell corresponds to one of the points. All the points in one cell are closer to the corresponding cell than to any other cell. This problem has an centralized optimal algorithm with O(nlogn) complexity [41], but the optimal algorithm is not applicable to distributed systems. In a distributed system, computing the *Voronoi* diagram where the nodes are modeled as points in the plane leads to a challenging problem. This is because, in some distributed systems, a node cannot gather

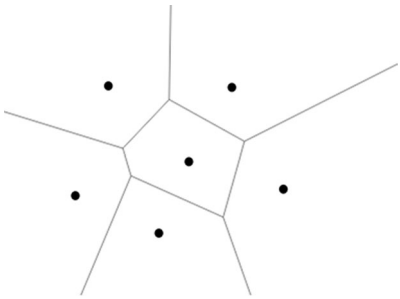


Fig. 2 An example for *Voronoi* diagram for six points in a plane

information about all nodes of the system and the *Voronoi* construction algorithms for building the *Voronoi* diagram among the nodes use information of all nodes. In the next paragraph, we focus on a type of *Voronoi* construction algorithms reported for distributed systems.

Several distributed algorithms are reported in [42–45] for constructing *Voronoi* diagrams in a distributed manner. In [42, 44], a type of algorithms is reported for constructing *Voronoi* diagram in a distributed manner for sensor networks. In this type of algorithms, each node of the network uses information about 1-hop neighbors to build an initial local *Voronoi* cell. Then the node modifies it corresponding *Voronoi* cell gradually as it gathers messages containing location information from other neighboring nodes.

3 Closed asynchronous dynamic cellular learning automaton with varying number of learning automata in each cell (*CADCLA-VL*)

A *CADCLA-VL* is a network of cells whose structure changes with time and each cell contains a set of *LAs* and a set of attributes. In this model, the connections among cells, and the number of *LAs* of each cell changes during the course of evolution of *CLA*. This model can be formally defined by 9 tuples as follows

$$CADCLA{-}VL = (G, A, \Psi, \Phi, N, F_1, F_2, F_3, F_4),$$

where

- $G = (V, E)$ is an undirected graph which determines the structure of *CADCLA-VL* where $\mathrm{V} = \{cell_1, cell_2, \ldots, cell_n\}$ is the set of vertices and E is the set of edges.
- $\mathrm{A} = \{LA_1, LA_2, \ldots, LA_m\}$ is a set of *LAs*. A subset of set A is assigned to a cell.
- $\mathrm{N} = \{\mathrm{N}_1, \mathrm{N}_2, \ldots, \mathrm{N}_n\}$ 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 $cell_i$ which constitute its local environment.
- $\Psi = \{\Psi_1, \Psi_2, \ldots, \Psi_n\}$, $\Psi_i = \{(j, X_j, C_j) | cell_j \in N_i\}$ denotes the *attribute* of $cell_j$ where $X_j \subseteq \{x_1, x_2, \ldots, x_s\}$ and $C_j \subseteq A \cdot \{x_1, x_2, \ldots, x_s\}$ is the set of allowable *attributes*. Ψ_i^1 determines the *attribute* of $cell_i$ when $\theta_i = 1$.

- $\Phi = \{\Phi_1, \Phi_2, \ldots, \Phi_n\}$ where $\Phi_i = \{(j, k, \alpha_l) | cell_j \in N_i \text{ and action}\alpha_l \text{ has been chosen by } LA_k \in C_i\}$ denotes the *state* of $cell_i$. $\Phi_i^!$ determines the *state* of $cell_i$ when $\theta_i = 1$.
- $F_1 : (\underline{\Psi}) \rightarrow (\underline{\zeta})$ is the *restructuring function*. In each cell, the *restructuring function* computes the *restructuring signal* based on the *attributes* of the cell and its neighboring cells. For example, in $cell_i$, the *restructuring function* takes $\langle \Psi_i \rangle$ and then returns a value from the closed interval [0, 1] for ζ_i^1 which is the *restructuring signal* of $cell_i$. We define $\zeta_i = \{(j, \zeta_j^1) | cell_j \in N_i)\}$ as the set of restructuring signals of neighbors of $cell_i$. In a cell, depending the application the value of the *restructuring signal* determines whether the neighbors of that cell should be changed or not. If the ζ_i^1 is equal to zero (one) this means that the neighbors of $cell_i$ are appropriate (not appropriate).
- $F_2 : (\underline{N}, \underline{\Psi}, \underline{\zeta}) \rightarrow (\underline{N}^1, \underline{\Psi}^1)$ is the *structure updating rule*. In each cell, the *structure updating rule* finds the immediate neighbors and *attribute* of the cell based on the *restructuring signal* computed by the cell, the *attributes* of the neighbors of the cell, and the neighbors of the cell.
- $F_3 : (\underline{\zeta}) \rightarrow (\underline{\upsilon})$ is the *automata trigger function*. Upon activation of a cell, *automata trigger function* is called to determine whether the learning automata residing in that cell to be triggered or not. If the *automata trigger function* returns true then the learning automata of the cell will be triggered. In $cell_i$, the *automata trigger function* takes $\langle \zeta_i \rangle$ and returns a value from {true, false} for v_i where v_i is called *automata trigger signal*. In a cell, since the value of the *restructuring signal* affects on the changes in the composition of the neighboring cells of that cell, the value of the *restructuring signal* is used to determine the value of the *automata trigger signal*. In $cell_i$, If the v_i is equal to true (false) then the learning automata of $cell_i$ are triggered (not triggered).
- $F_4 : (\underline{\Phi}, \underline{\Psi}) \rightarrow (\underline{\beta})$ is the *local rule* of *CADCLA-VL*. In each cell, the *local rule* computes the *reinforcement signal* for the learning automata of that cell based on the *states* and the *attributes* of that cell and its neighboring cells. For example, in $cell_i$, *local rule* takes $\langle \Phi_i, \Psi_i \rangle$ and then computes the *reinforcement signal* $\langle \beta_i \rangle$ for the learning automata of $cell_i$.

The order by which the cells of *CLA* will be activated is application dependent. Upon the activation of a cell, the cell performs a process which has three phases: *preparation*, *structure updating* and *state updating*. The pseudo code of this process is given in Fig. 3. These three phases are described bellow.

1. *Preparation phase* In this phase, a cell performs the following step

   Step 1: The cell and its neighboring cells compute their *restructuring signals* using the *restructuring function* (F_1).

Algorithm Activate ()

Input: $cell_i$

Notations: F_1 denotes the *restructuring function*
F_2 denotes the *structure updating rule*
F_3 denotes the *automata trigger function*
F_4 denotes the *local rule*
Ψ_i denotes the *attribute* of $cell_i$
Φ_i denotes the *state* of $cell_i$
ζ_i^1 denotes the *restructuring signal* of $cell_i$
ν_i denotes the *automata trigger 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//

Compute ζ_i^1 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^1 is 1) **Then**
Compute N_i and Ψ_i using F_2;
EndIf
Call F_3 to determine the value of ν_i;

//state updating phase//

If (ν_i is true) **Then**
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_4;
Update the action probabilities of *learning automata* of $cell_i$ using β_i;
EndIf
End

Fig. 3 The pseudo code of the process which is executed upon the activation of a cell

2. *Structure updating phase* In this phase, a cell performs the following steps

Step 1: 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.
Step 2: The *automata trigger function* (F_3) whether the set of learning automata of the cell must be triggered or not depending on the *restructuring signal*.
Step 3: If the learning automata are triggered then the cell goes to the *state updating phase*.

Step 4: If the set of learning automata of the cell are not triggered then the activation process terminates.

3. *State updating phase* In this phase, a cell performs the following steps

Step 1: Each learning automaton of the cell selects one of its actions. The set of actions selected by the set of learning automata in the cell determines the new *state* for that cell.
Step 2: The *local rule* (F_4) is applied and a *reinforcement signal* is generated according to which the action probability vectors of the learning automata of the cell are updated.

The internal structure of $cell_i$ and its interaction with local environments are shown in Fig. 4. In this model, each cell has a set of learning automata which may vary during the operation of *CLA* and three components: *restructuring signal generator*, *structure updater and automata trigger function* as explained below.

- *Restructuring signal generator* This unit generates the *restructuring signal* using the *restructuring function*.
- *Structure updater* This unit updates the set of neighboring cells of the cell according to the *restructuring signal* and *structure updating rule*.
- *Automata trigger function* This unit determines whether the set of *LAs* of the cell to be triggered or not according to the *restructuring signal*.

Performance of *CADCLA-VL* will be studied using two metrics: *entropy*, and *potential energy*, which are described in the rest of this section.

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

$$A(t) = \sum_{i=1}^{n} v_i(t) \tag{7}$$

where $v_i(t)$ is the *restructuring signal* of $cell_i$ at iteration t.

Potential energy can be used to study the changes in the structure of *CLA* as it interacts with the environment. If the value of $A(t)$ becomes zero then no further change needs to be made to the structure. A higher value of $A(t)$ 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 $cell_l$ of the *CLA*. The *entropy* of $cell_l$ is defined by Eq. (9)

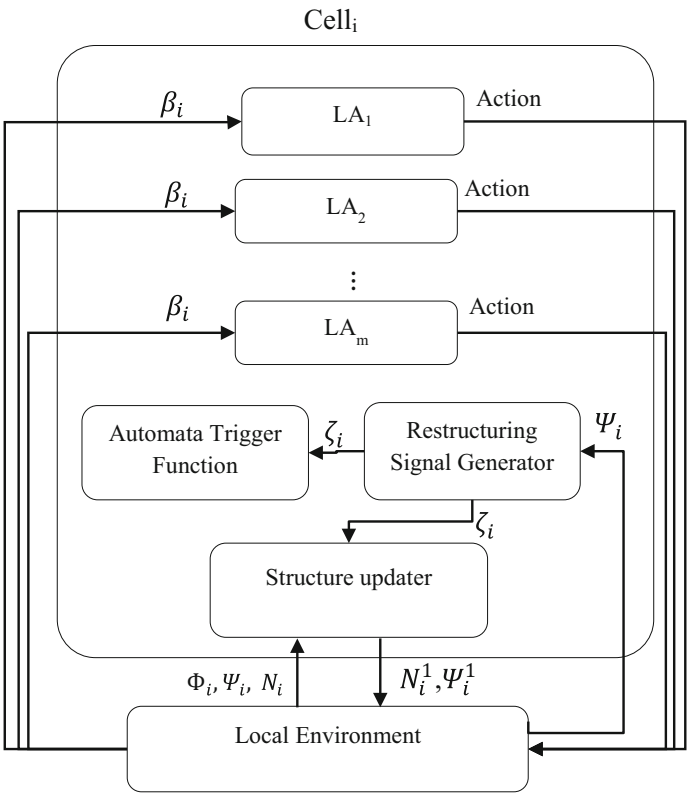


Fig. 4 Internal structure of $cell_i$ and its interaction with local environment

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

where $H_i(t)$ is the *entropy* of learning automaton LA_i and C_l (t) denotes the set of indexes of learning automata of $cell_l$ at iteration t. $H_i(t)$ is the *entropy* of learning automaton LA_i defined by Eq. (10),

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

where r_i is the number of actions of the learning automaton LA_i. p_{ij} (t) is the probability of selecting action α_j of learning automaton LA_i at iteration t.

Entropy of the *CLA* can be used to study the changes occurring in the *states* of the cells *of CLA*. The value of zero for $H(t)$ means that the learning automata of the cells 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 A CLA based landmark clustering algorithm for unstructured peer-to-peer networks: *xOverlay*

Initially, a *CLA* isomorphic to the overlay graph is created which involves defining the initial structure, *local rule*, *structure updating rule*, *automata trigger function*, *restructuring function*, and *local environments*. The overlay graph is then mapped into the *CLA* in which each cell is equipped with a learning automaton. Each peer plays one of three roles: *Undecided*, *Ordinary* and *Landmark*. The initial role of each peer is set to *Undecided*. The learning automata in each cell have two actions "change the role to Landmark" and "change the role to Ordinary". We call the set of nodes in each cell a cluster of peers. Clearly, at the beginning of the algorithms each cluster contains one peer of the overlay graph and hence the number of clusters is equal to the number of cells in the *CLA*. Each cell of the *CLA* has an *attribute* which contains the geographical positions of the peers of the cell in the underlay graph. Figure 5 shows a snapshot of the *CLA* at a particular time. As shown, $cluster_i$, $cluster_j$, and $cluster_k$ are clusters corresponding to $cell_x$, $cell_y$, and $cell_z$ respectively. $peer_i$, $peer_j$, and $peer_k$ are the landmark peers of $cluster_i$, $cluster_j$, and $cluster_k$ respectively. Once the *CLA* is created, the proposed algorithm utilizes it to manage the roles of the peers. The process executed by each peer $peer_i$ when joining to the network consists of three phases: *Construction* phase, *Organization* phase, and *Maintenance* phase (Fig. 6). These phases are briefly described as below.

- *Construction* phase: During the *Construction* phase performed by a peer, the peer tries to find an appropriate cluster and sets an initial role for itself. If the peer cannot find any cluster, it sets its role to *Landmark*, and goes to *Maintenance* phase. If the peer found a cluster, it connects to the cluster, sets its role to *Ordinary*, and goes to *Organization* phase.
- *Organization* phase: During the *Organization* phase performed by a peer, the peer and other peers which they have the same cluster as the cluster of that peer try to find an appropriate landmark peer for their cluster by activating their corresponding cell in the *CADCLA-VL*. In a cell, the changes were made by the activation function in the *CADCLA-VL* may change the connections of the peer and also select a new landmark peer according to feedbacks received from the network. In other word, as the algorithm proceeds, the structure of *CLA* changes and as a result the number of peers in a cell may change. At the end of *Organization* phase, the peer goes to the *Maintenance* phase.
- *Maintenance* phase: In this phase, the peer monitors (and broadcast) required information from (to) all peers of its corresponding cluster, and then execute appropriate management operation. In this phase, the peer may jump to *Organization* or *Construction* phases in some situations.

Now, we complete the description of the algorithm by describing the *Local environment*, *Automata trigger function*, *Structure updater*, and *Restructuring signal generator* for the *CLA* used by function *Activate (cell)* called by Algorithm *xOverlay* given in Fig. 6. The detailed descriptions of this algorithm are given in Appendix 1.

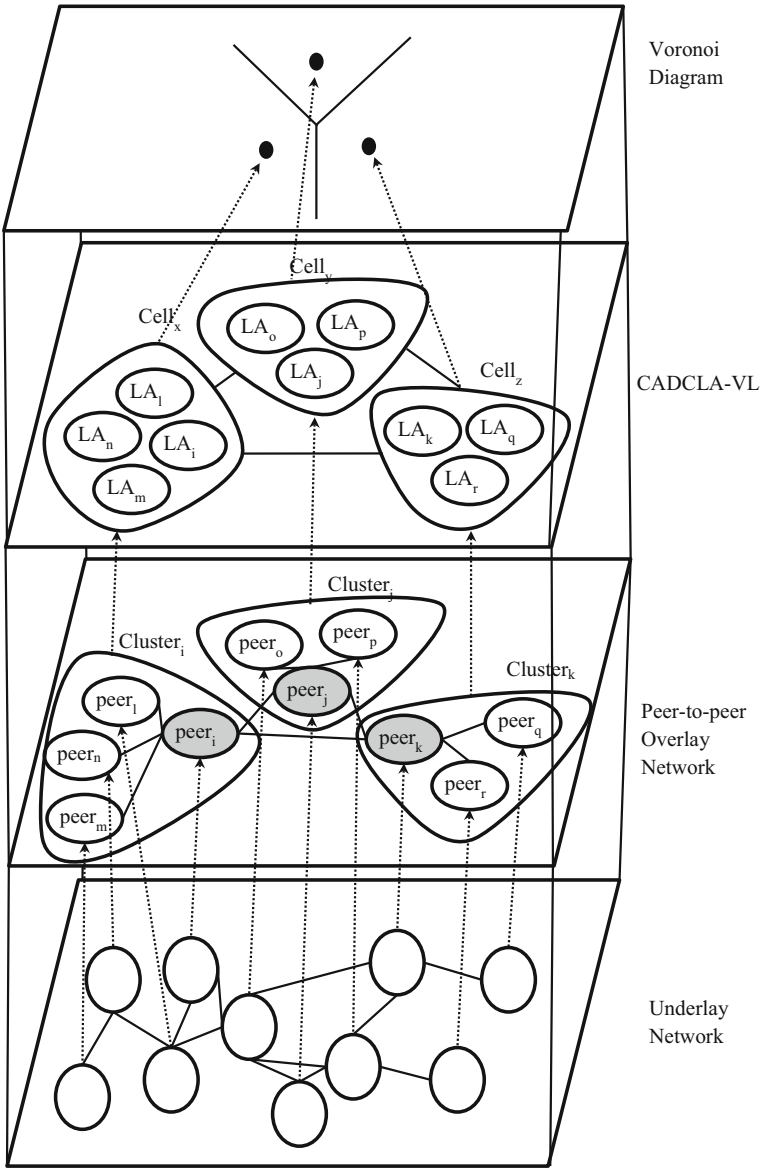


Fig. 5 A snapshot of the *CLA* at a particular time

- *Restructuring signal generator* The input and output of this unit are shown in Fig. 7. This unit takes information about neighbors of a cell as input and returns a *restructuring signal*. Let $cluster_i$ be the corresponding cluster to $cell_i$. The *restructuring signal* of $cell_i$ is set to 1 if there is a *departed* peer returned by the function *Voronoi_center_selector* for $cluster_i$, and 0 otherwise. The pseudo code of the *restructuring signal generator* is given in Fig. 8. In this pseudo code,

Algorithm *xOverlay* ()

Notation: $Cluster_x$ denotes the cluster that the *peer* belongs to at any time. *cell* denotes the cell corresponds to the *cluster* $Cluster_x$.

```
01  Begin
02                          //Construction phase//
03    If (there is an appropriate cluster cluster_x) Then
04      Connect to the Cluster_x;
05      Set the role of the peer to Ordinary;
06    Else
07      Set the role of the peer to Landmark;
08      goto Maintenance phase; // goto line 13//
09    EndIf
10                          //Organization phase//
11    Call Activate (cell) ;
        // Algorithm activate given in Fig. 3 is  used to activate the cell//
12    Determine the role of the peers according to the state of cell//
13                          //Maintenance phase//
14    While (management phase is needed) Do
15      Broadcast  management information of cluster cluster_x;
16      If (reorganizing in the cluster is needed) Then
17        Goto Organization phase; // goto line line 10//
18      EndIf
19      If (reconstruction in the cluster is needed) Then
20        Goto Construction phase; // goto line line 02//
21      EndIf
22    EndWhile
23  End
```

Fig. 6 Simplified pseudo code of the proposed algorithm

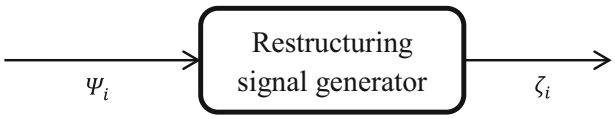


Fig. 7 Input and output of the restructuring signal generator of $cell_i$

function *Voronoi_center_selector* given in Fig. 9 is used to find *departed* peers. In this function, function *distance_evaluator* is used to determine whether two values of delays can be considered equal or not (Fig. 10). Function *distance_evaluator* and function *Voronoi_center_selector* are described in Remark 1 and Remark 2 respectively.

Remark 1 In a peer-to-peer network, the information about delays among peers cannot be gathered with high accuracy. Inaccurate information results in creating inaccurate *Voronoi* cells. We solved this problem using a function called *distance_tester* which is described as follows. Function *distance_tester* takes two values (m and n), and threshold $q \in [0, 1)$ as input and then returns true if $(m \geq n \text{ and } (1-q)\cdot m \leq n)$ *or* $(m < n \text{ and } (1-q)\cdot n \leq m)$ and false

```
Algorithm Restructuring signal generator
Input:      Attributes of a cell_i and its neighboring cells
Output:     Restructuring signal
Notations:  Let cluster_i be the set of nodes of cell_i.
            Let d-peers be the set of departed peers of cell_i.
Begin
 Find d-peers;// using function Voronoi_center_selector given in Fig. 9//
 If (d-peers is not empty) Then
    Return(1);
 Else
    Return(0) ;
 EndIf
End
```

Fig. 8 The pseudo code of the restructuring signal generator

```
Algorithm Voronoi_center_selector()
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. 10) 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. 9 The pseudo code for function *Voronoi_center_selector*

otherwise. This function determines whether two values of delays can be considered equal or not. This function is also used in landmark clustering algorithms such as those reported in [7, 34].

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 $(x \geq y \; and \; (1-r) \times x \leq y)$**or**
    $(x < y \; and \; (1-r) \times y \leq x)$ **Then**
      z ← True;
  Else
      z ← False;
  EndIf
  Return (z);
End

Fig. 10 The pseudo code for function *distance_ evaluator*

Remark 2 In each cluster, function *Voronoi_center_selector* categorizes the peers of that cluster and returns some information about that cluster. In $cluster_i$, this function takes a threshold t, the delays among all peers of $cluster_i$ and delays from all peers of $cluster_i$ to the landmark peers of the clusters adjacent to $cluster_i$ and then returns m, v, *c-peer*, and *d-peers*. *c-peer* is called *central candidate* peer and it has the minimum mean and variance of delays to the landmark peers of the clusters adjacent to $cluster_i$ and ordinary peers of $cluster_i$. In $cluster_i$, the *central candidate* peer is an appropriate landmark peer. m and v denote the mean and variance of delays computed for peer *c-peer* respectively. *d-peers* is called *departed* peers set which contains *departed* peers. In $cluster_i$, a $peer_d$ is called *departed* peer if the delays from $peer_d$ to the landmark peers of adjacent clusters of the $cluster_i$ is not equal (based on the output of function *distance_tester*) to the delays from the *c-peer* to the landmark peers of the adjacent clusters of the $cluster_i$. The algorithm used for finding *central candidate* peer is similar to an algorithm used for finding the center point of *Voronoi* cells reported in [42, 44].

- *Automata Trigger Function* The input and output of this unit are shown Fig. 11. This unit takes the *restructuring signal* of a cell as input and then returns true or false which determines whether the learning automata of the cell are activated or not. This unit returns true if the *restructuring signal* of the cell is equal to 0 and false otherwise. The pseudo code for this unit is given in Fig. 12.
- *Local environment* The input and output of this unit are shown in Fig. 13. The *reinforcement signal* β_i is computed by applying the *local rule*. This unit returns 1, if the learning automaton of $peer_j$ (returned by the *Voronoi_center_selector* function as *central candidate* peer) has selected "set the role to landmark peer",

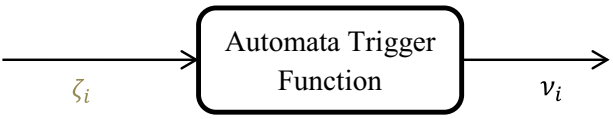


Fig. 11 Input and output of the automata trigger function of $cell_i$

Algorithm Automata Trigger Function
Input: *restructuring signal* of $cell_i$
Output: *automata trigger signal*
Begin
If (the *restructuring signal* is equal to 0)**Then**
Return (True);
Else
Return (False);
EndIf
End

Fig. 12 The pseudo code of the automata trigger function

Fig. 13 Input and output of the local environment of $cell_i$

other learning automata of $cell_i$ have selected "set the role to ordinary peer" and there is no departed peer (returned by the *Voronoi_center_selector* function as *d-peer*), and 0 otherwise. The pseudo code for this unit is given in Fig. 14.

- *Structure updating rule* The input and output of this unit are shown in Fig. 15. This unit is implemented using three operations called *Migrate* operation, *split* operation, and *Merge* operation which are described in Fig. 16. This rule use the concept of *Voronoi* diagrams to appropriately divide the peers among the clusters in such a way that an appropriate clustered network with low communication delay can be obtained.

1. *Migrate operation* Figure 17 shows an example of the *Migrate* operation. In this figure, $cell_i$ and $cell_j$ have participated in a *Migrate* operation. Let $cluster_i$ and $cluster_j$ denote the clusters of peers for $cell_i$ and $cell_j$ respectively and $peer_i$ and $peer_j$ are the landmark peers of $cluster_i$ and $cluster_j$, respectively and also $peer_k$ is returned by function *Voronoi_center_selector* as a *departed* peer for $cluster_i$. In $cluster_i$, the *structure updating rule* of $cell_i$ uses *Migrate* operation to assign

Algorithm Local rule

Input: *attribute* and *state* of $cell_i$
Output: *reinforcement signal*
Notations: Let $cluster_i$ be the set of nodes of $cell_i$.
Let *c-peers* be the set of candidate peers of $cell_i$.
Let *d-peers* be the set of departed peers of $cell_i$.

Begin
Find *c-peer*;// using fuction *Voronoi_center_selector* given in Fig. 9//
Find *d-peer*;// using fuction *Voronoi_center_selector* given in Fig. 9//
If ((the learning automaton of a *c-peer* has selected "*set the role to landmark peer*") **and** (the other learning automata of $cell_i$ have selected "*set the role to ordinary peer*") **and** (*d-peer* is empty))**Then**
Returns(1);
Else
Returns(0);
EndIf
End

Fig. 14 The pseudo code of the local environment

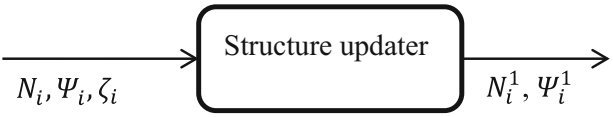


Fig. 15 Input and output of the structure updater of $cell_i$

the LA_k (the learning automaton of $peer_k$) of $cell_i$ to $cell_j$. $cell_j$ is a neighboring cell of $cell_i$ whose landmark peer has the least delay to the *landmark* peer of $cell_i$. Note that, assigning the LA_k to $cell_y$ leads to changing the connection of $peer_k$ from $peer_i$ to $peer_j$.

2. *Split operation* Figure 18 shows an example of the *Split* operation. In this figure, $cell_i$, $cell_j$, $cell_w$, and $cell_z$ have participated in a *Split* operation. Let $cluster_i$ and $cluster_j$ are the clusters of peers for $cell_i$ and $cell_j$ respectively, and $peer_j$ is returned by function *Voronoi_center_selector* as a *departed* peer for $cluster_i$. In $cluster_i$, if $node_i$ cannot find an appropriate cell for executing the *Migrate* operation with it, the *structure updating rule* of $cell_i$ uses *Split* operation to split the $cell_i$ into two cells; a new cell which contains the learning automaton of $peer_j$ and a cell which contains the rest of learning automata of $cell_i$. Note that the neighbors of the new cell are the neighbors of $cell_i$ (before execution of *Split* operation).
3. *Merge operation* Figure 19 shows an example of the *Merge* operation. In this figure, $cell_a$, $cell_b$, $cell_i$ and $cell_j$ have participated in a *Merge* operation. The *Merge* operation is performed if there is no learning automaton in $cell_i$ ($cell_j$) after the execution of *Migrate* operation.

Algorithm Structure updater

Input: neighbors, *attributes* and *restructuring signals* of $cell_i$ and its neighbors

Output: Immediate neighbors and *attribute* of $cell_i$

Notations: Let $cluster_i$ be the set nodes of $cell_i$.
Let $cluster_j$ 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-peers;// using function Voronoi_center_selector given in Fig. 9//
    If (d-peers is not empty) Then
      Select a peer peer_k from set d-peers randomly;
      Select an appropriate cluster cluster_j for peer_k;
      If (cluster_j has been found) Then
        If (cluster_i has one peer) Then
          Execute Merge operation;
        Else
          Execute Migrate operation;
        EndIf
      Else
      Execute Split operation;
    EndIf
  EndIf
EndIf
End
```

Fig. 16 The pseudo code of the structure updater

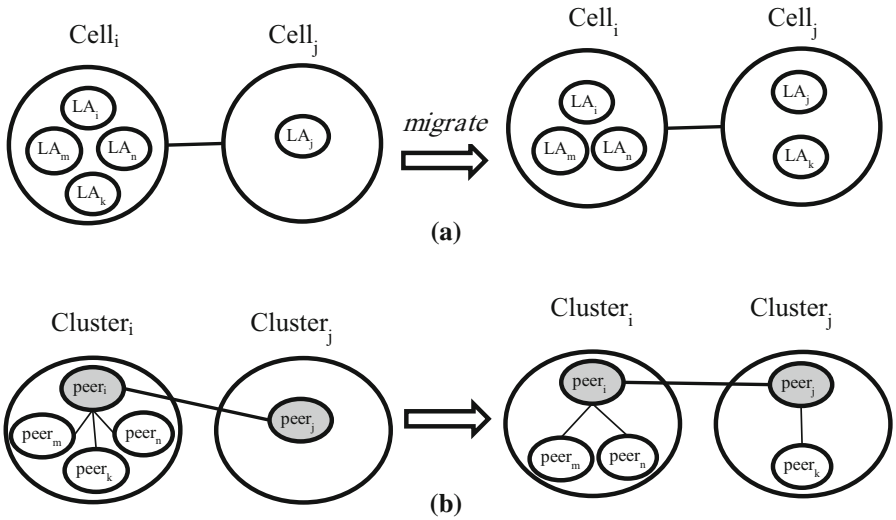


Fig. 17 **a** Example of *migrate* operation for two cells. **b** The effect of migrate operation on the structure of the peer-to-peer overlay network

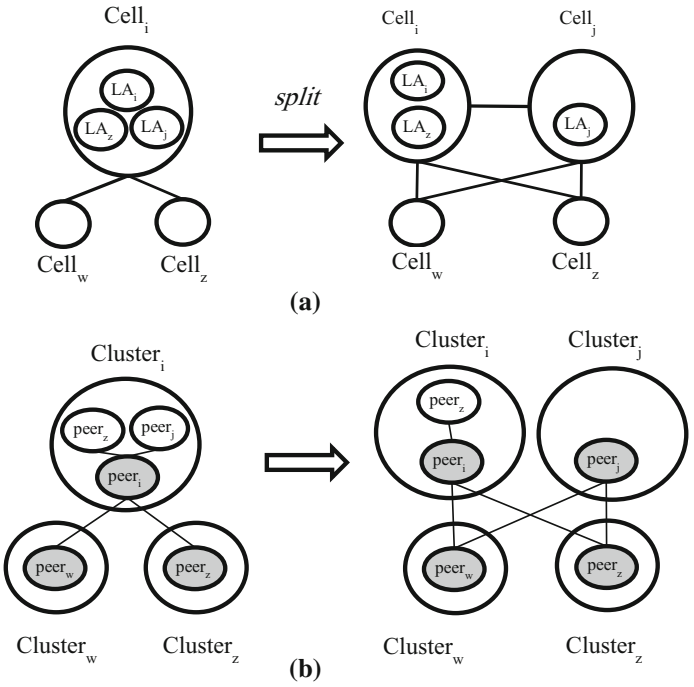


Fig. 18 **a** Example of *split* operation. **b** The effect of *split* operation on the structure of the peer-to-peer overlay network

5 Experimental results

In this section, computer simulations have been conducted to evaluate the performance of the proposed algorithm (*xOverlay*) and then the results are compared with the results obtained for other existing algorithms. All simulations have been implemented using the *OverSim* which is a flexible overlay network simulation framework [46]. Different types of underlay network models such as *Simple* and *ReaSE* are supported in *Oversim*. The *Simple model* is the most scalable model and the *ReaSE model* is able to generate different types of underlay networks. We used the *ReaSE model* to generate router-level topologies [47]. We used two underlying network topologies: *T*.1 (*k*) as an example of the *Simple model* where *k* determines the size of the network, and *T*.2, *T*.3, *and T*.4 as examples of the *ReaSE model*. The detailed descriptions of these topologies are given below.

- T.1 (k): In this underlay topology, k peers are placed on a N-Dimensional Euclidean space and the Internet latencies are based on CAIDA/Skitter data [48, 49].
- T.2: Consists of 10 autonomous systems, and about 1000 router-level peers.
- T.3: Consists of 50 autonomous systems, and about 200 router-level peers.
- T.4: Consists of 100 autonomous systems, and about 100 router-level peers.

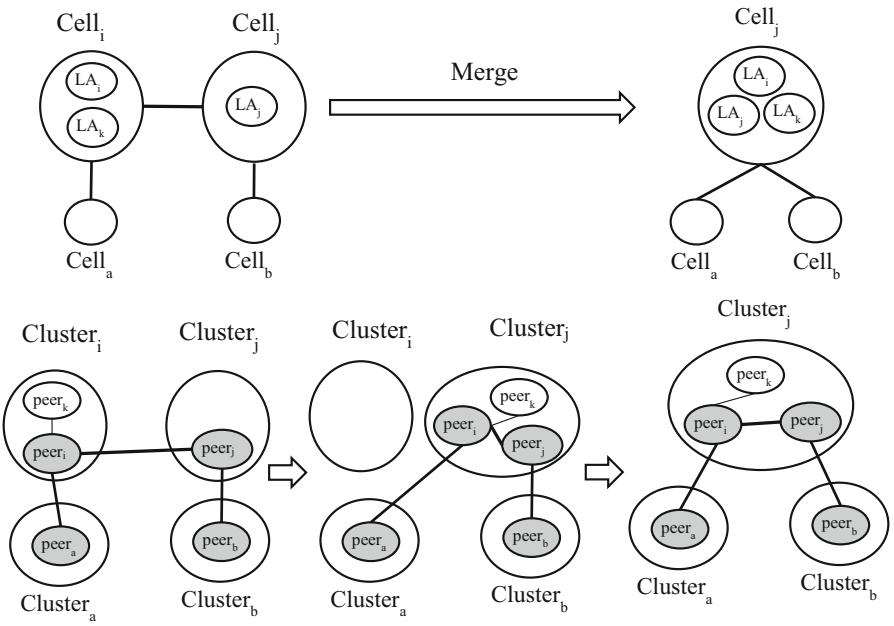


Fig. 19 **a** Example of *merge* operation. **b** The effect of *merge* operation on the structure of the peer-to-peer overlay network

To evaluate the performance of the *xOverlay algorithm*, it was compared with two groups of algorithms. The first group contains the algorithms reported in the literature for solving the landmark clustering problem listed below.

- The *mOverlay* algorithm [25] which is a well-known landmark clustering algorithm in unstructured peer-to-peer networks.
- The *lOverlay* algorithm [7] reported recently is an extension of the *mOverlay* algorithm.

The second group contains different versions of the *xOverlay algorithm*. These algorithms are used to study different aspects of the proposed learning model used in *xOverlay algorithm*. These visions are briefly described below.

- The *nOverlay* algorithm is a version of the *xOverlay algorithm* in which *CADCLA-VL* is replaced with a *pure-chance CADCLA-VL*. A *pure-chance CADCLA-VL* is a *CADCLA-VL* in which, each learning automaton is replaced with a *pure-chance* automaton. In this case, each peer participates in the landmark selection procedure with probability 0.5. *nOverlay* will be used to study the learning capability of the *CLA* on the performance of *xOverlay algorithm*.
- The *hOverlay* algorithm is a version of the *xOverlay algorithm* in which the learning automata of *CADCLA-VL* are deactivated. In this algorithm, each peer always participates in the landmark selection procedure without considering the

actions selected by the learning automata. This means that restructuring of the network is performed based on the positions of the peers only. This is in contrast to *xOverlay* in which the restructuring of the network depends on both the positions of the peers and the actions selected by the learning automata. *hOverlay* was used to study the performance of the landmark clustering algorithm when the learning automata are absent.

- *kOverlay* (*p*) algorithm is a version of *xOverlay* in which the information about each peer such as its position and the actions selected by the learning automata residing the cells containing that peer is used with probability p. *kOverlay* (*p*) was used to study the effect of the above mentioned information on the performance of *xOverlay* algorithm.

The *lOverlay* algorithm has four main parameters: *a*, *t*, *MAX_ITERATION_LEARNING*, and *MAX_ITERATION*. Based on our experimental study using different parameter settings, we have chosen the best parameters values for *lOverlay* algorithm. The parameters of the *lOverlay* algorithm are set according to Table 1. The *xOverlay* algorithm has two main parameters: *t*, and *THRSHOLD*. Parameter *THRSHOLD* must be set to a large value in order to provide enough time for the peers of the network to communicate with their neighbors. For the experiments, *THRSHOLD* is set to 5 s. For all experiments, each peer is equipped with a variable structure learning automaton of type L_{RI} with reward parameter $a = 0.1$ and the neighborhood radius of all cells of the *CADCLA-VL* is set to 2.

The results reported are averages over 60 different runs. In each run, the algorithms are executed for 100 rounds. The algorithms are compared with respect to six metrics: Total Communication Delay (*TCD*), Mean Round-trip Time (*MRT*), Control Message Overhead (*CMO*), *entropy*, and Potential Energy (*PE*). The definitions of *entropy*, *PE* were previously given in Sect. 4. The definitions of other metrics are given below.

- *Total Communication Delay* is the total of all-pairs end-to-end communication delay of the overlay network according to Eq. (3). The aim of any landmark clustering algorithm is to reduce the total communication delay as much as possible.
- *Mean Round-trip Time* is the mean round-trip time between members of the same cluster. Clusters with only one peer are ignored in this case.
- *Control Message Overhead* is the number of extra control messages generated for the purpose of reconfiguration of overlay network.

Table 1 The parameters of *lOverlay* algorithm

Parameters	Values
a	0.1
t	0.3
MAX_ITERATION_LEARNING	1
MAX_ITERATION	1000

A summary of the experiments conducted in this paper is as follows. Experiment 1 is designated to study the impact of parameter q on the performance of the *xOverlay* algorithm. Experiment 2 is conducted to study the impact of the learning capability of the learning automata of the *CADCLA-VL* on the performance of the *xOverlay* algorithm with respect to *TCD*, *MRT* and *CMO*. Experiment 3 is designated to study the performance of the *CADCLA-VL* of the *xOverlay* algorithm with respect to *entropy*, *and PE*. Experiment 4 and experiment 5 are conducted to study the effect of network size and underlay topology on the *mOverlay*, the *lOverlay* and the *xOverlay* algorithm.

- Experiment 1

This experiment is conducted to study the impact of the parameter q on the performance of the *xOverlay* algorithm. For this purpose, the *xOverlay* algorithm is tested for five values for parameter $q = 0.1, 0.2, 03, 0.4$ and 0.5. In this experiment, *T*.1 (10,000) is used as the underlay topology. The results are compared with respect to *TCD*, *MRT*, and *CMO*. According to the results of this experiment that are given in Table 2, we may conclude that increasing parameter q leads to increasing *MRT* and *CMO*. In terms of *TCD*, *xOverlay* algorithm performs well for $q = 0.3$.

- Experiment 2

This experiment is conducted to study the impact of the learning capability of the learning automata of the *CADCLA-VL* on the performance of *xOverlay* algorithm with respect to *TCD*, *MRT* and *CMO*. For this purpose, *xOverlay* algorithm is compared with *nOverlay* and *hOverlay* algorithms. For this experiment parameter q is set to 0.3, and *T*.1 (10,000) is used as underlay topology. The results are compared with respect to *TCD*, *MRT*, *and CMO*. According to the results of this experiment given in Table 3, we may conclude the followings.

- In terms of *TCD*, and *MRT*, *xOverlay* algorithm performs better than *nOverlay* because in *xOverlay* algorithm each peer decides on its role (to be an ordinary or landmark) adaptively based on the decisions made by its neighboring peers whereas in *nOverlay* at a given time the history of a peer about its role does not affect its decision regarding its role (chooses its role with probability 0.5).
- In terms of *TCD*, and *MRT*, the *xOverlay* algorithm performs better than *hOverlay*. This is because in *hOverlay*, unlike *xOverlay* all the learning automata are deactivated and as a result each peer always participates in the landmark selection procedure. In *hOverlay* restructuring of the network is performed based on the positions of the peers only whereas in *xOverlay* restructuring of the network depends on both the positions of the peers and the actions selected by the learning automata.
- The *xOverlay* algorithm has lower *CMO* as compared to *hOverlay* algorithm. This is because in *hOverlay*, each peer always participates in the landmark clustering which results in generating higher number of control messages. *xOverlay* has lower *CMO* as compared to *nOverlay* algorithm. This is because

Table 2 The impact of the parameter *q* on the performance of *xOverlay*

	Parameter q				
	q = 0.1	q = 0.2	q = 0.3	q = 0.4	q = 0.5
TCD	176,881 ± 812 ms	174,782 ± 514 ms	173,891 ± 213 ms	173,954 ± 503 ms	179,941 ± 913 ms
MRT	5.8 ± 0.69 ms	5.89 ± 0.49 ms	6.6 ± 0.21 ms	8.4 ± 0.79 ms	10.4 ± 1.12 ms
CMO	110,152 ± 15,109	130,897 ± 9201	170,019 ± 8931	250,210 ± 19,201	342,102 ± 21,032

Table 3 Comparison of different versions of the proposed algorithm

	xOverlay	hOverlay	nOverlay
TCD	173,980 ± 223 ms	174,300 ± 314 ms	179,980 ± 1158 ms
MRT	6.6 ± 0.21 ms	6.7 ± 0.34 ms	8.4 ± 0.41 ms
CMO	170,113 ± 2952	191,232 ± 1521	181,132 ± 2115

nOverlay performs an exhaustive search in order to find appropriate landmarks whereas the search performed by *xOverlay* is guided by *CLA*.

- Experiment 3

This experiment is conducted to study the performance of the *CADCLA-VL* of *xOverlay* algorithm with respect to *entropy*, *and PE*. For this purpose, *xOverlay* algorithm is compared with *nOverlay* and *kOverlay* (0.8) algorithms. For this experiment parameter q is set to 0.3, and *T*.1 (10,000) is used as underlay topology. The results obtained from different rounds of the simulation are reported in Figs. 20 and 21. According to the results of this experiment, one may conclude the followings.

- In terms of *entropy*, *xOverlay* performs better than other algorithms. Figure 20 indicates that the *entropy* is high at initial rounds of the simulation and gradually decreases. This means that the changes in the *states* of *CLA* becomes less frequent as the set of landmark peers found by the algorithm becomes closer to the appropriate set of landmark peers.
- Figure 21 shows the changes in *PE* during the execution of *xOverlay*. This figure shows that *PE* is initially high and approaching zero indicating the fact that the overlay network topology is approaching to a fixed structure.

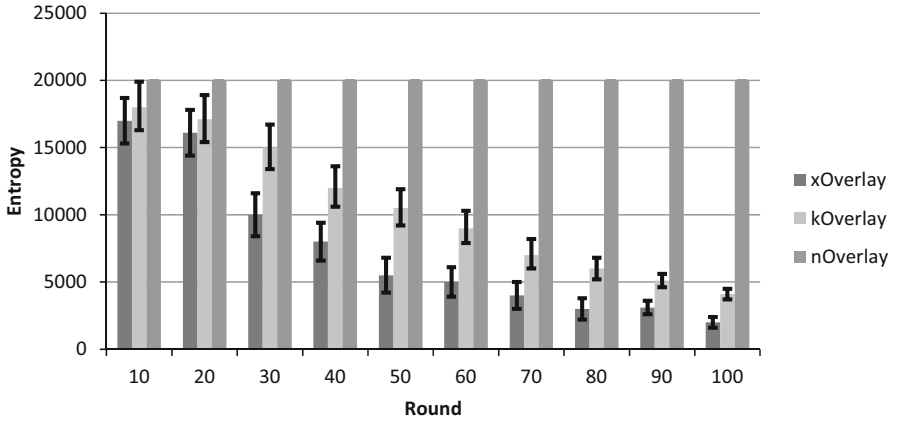


Fig. 20 Comparison of *xOverlay*, *kOverlay* (0.8), and *nOverlay* with respect to *Entropy*

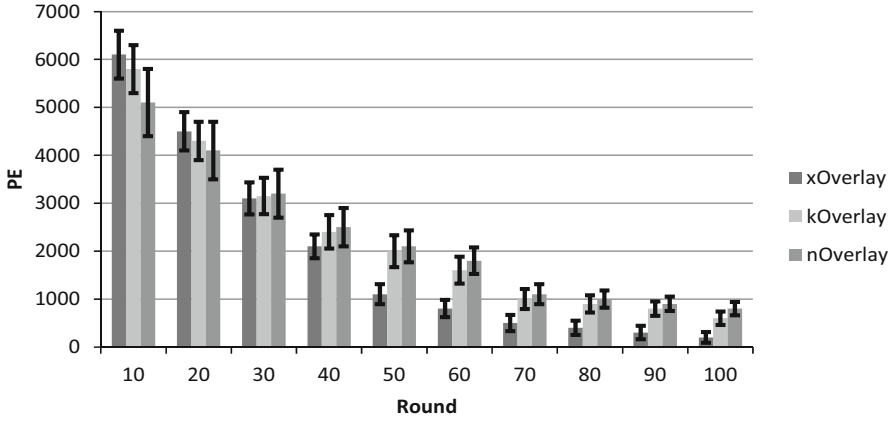


Fig. 21 Comparison of *xOverlay*, *kOverlay* (0.8), and *nOverlay* with respect to *PE*

Experimentations have shown that for *nOverlay* and *kOverlay*, *PE* shows a gradual decrease but approaching to a fixed structure is very slow.

- Experiment 4

This experiment is conducted to study the impact of the network size on the performance of the *xOverlay* algorithm when the underlay network topology is generated by the simple model as described before. For this purpose, we generate five topologies from *T*.1 (*k*) for $k = 10{,}000$, 20,000, 30,000, 40,000 and 50,000. For *xOverlay* algorithm, the parameter q is set to 0.3. The results obtained are compared with the results obtained for *mOverlay* and *lOverlay* algorithms with respect to the criteria mentioned above. According to the results of this experiment that are shown in Figs. 22, 23, and 24, one may conclude that in terms of *TCD* and *MRT*, *xOverlay* algorithm performs better than other algorithms because in *xOverlay* algorithm, *CADCLA-VL* gradually found appropriate landmark peers for the clusters which results in lower *TCD*, and *MRT*. *mOverlay* performs better than other algorithm in terms of *CMO* but worse than other algorithms in terms of *TCD* and *MRT*. This is because *mOverlay* unlike other algorithms is not adaptive and does not have the burden of generating messages for the sake of overlay structure adaptation.

- Experiment 5

In this experiment, we study the performance of *mOverlay*, *lOverlay* and *xOverlay* algorithms on topologies *T*.2, *T*.3, and *T*.4 which all of them belong to the class of router level topologies [47]. The results obtained from the *xOverlay* algorithm are compared with the results obtained for *mOverlay* and *lOverlay* algorithms with respect to *TCD*, *MRT*, and *CMO* when parameter q is set to 0.3. From the results of this experiment given in Tables 4, 5, and 6, we may conclude that the *xOverlay* algorithm performs better than *mOverlay* and *lOverlay* algorithms

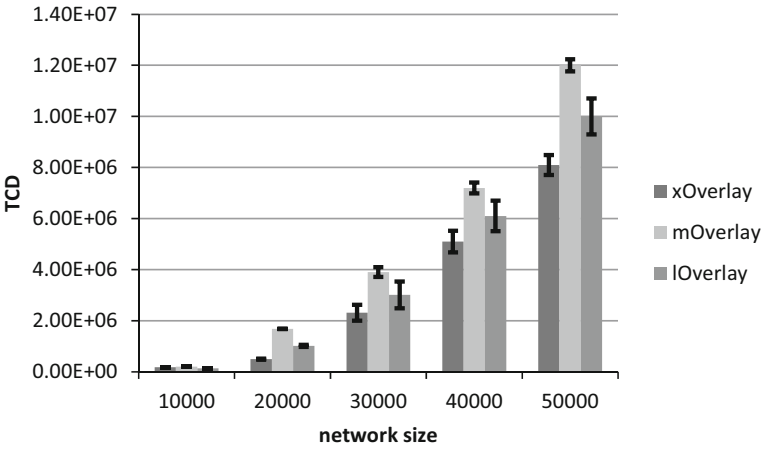


Fig. 22 Comparison of *xOverlay* with *mOverlay* and *lOverlay* with respect to *TCD*

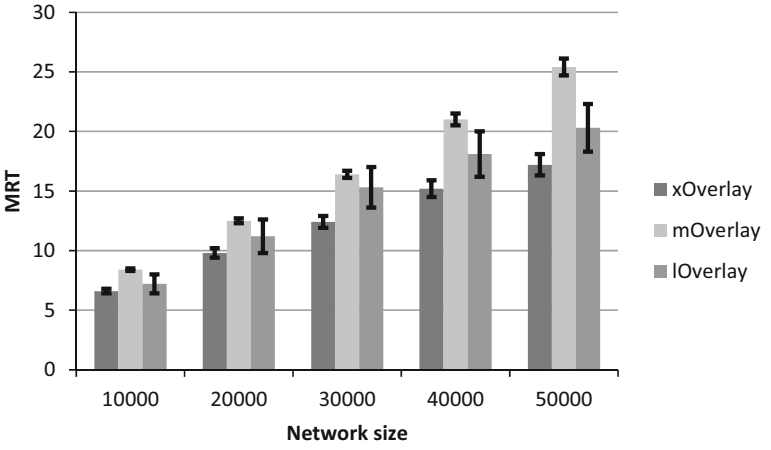


Fig. 23 Comparison of *xOverlay* with *mOverlay* and *lOverlay* with respect to *MRT*

in terms of *TCD* and *MRT*. This means that the proposed *CLA* based algorithm can be efficient even for router level topologies.

5.1 Summary of the results and discussion

In this section, a summary of the results and discussions about the performance of *xOverlay* algorithm in comparison with *mOverlay* and *lOverlay* algorithms are given as below.

- For *xOverlay* algorithm unlike other algorithms we use two metrics *entropy* and *Potential Energy* (*PE*) to study the adaptation process in the overlay network.

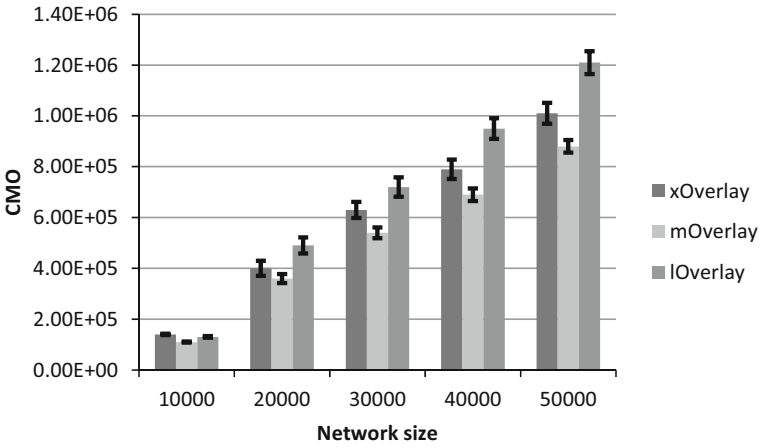


Fig. 24 Comparison of *xOverlay* with *mOverlay* and *lOverlay* with respect to *CMO*

Table 4 The impact of the underlay topology on the performance of *mOverlay*, *lOverlay* and the *xOverlay* with respect to *TCD*

	Topology		
	Topology T.2	Topology T.3	Topology T.4
lOverlay	197,542 ± 14,185 ms	167,780 ± 10,105 ms	169,794 ± 9311 ms
mOverlay	240,164 ± 10,058 ms	197,880 ± 6258 ms	227,961 ± 5201 ms
xOverlay	190,145 ± 12,124 ms	138,668 ± 8111 ms	148,792 ± 7162 ms

Table 5 The impact of the underlay topology on the performance of *mOverlay*, *lOverlay* and the *xOverlay* with respect to *MRT*

	Topology		
	Topology T.2 (ms)	Topology T.3 (ms)	Topology T.4 (ms)
lOverlay	23.1 ± 1.9	21.4 ± 1.8	18.2 ± 1.4
mOverlay	25.6 ± 1.3	22.3 ± 1	19.7 ± 0.8
xOverlay	19.5 ± 1.1	18.3 ± 0.7	15.7 ± 0.5

- The quality of the overlay obtained by the *xOverlay* algorithm with respect to *TCD* and *MRT* is higher than *mOverlay* and *lOverlay* algorithms for different network size and underlay topologies. Since the overlay network is used to transfer messages, finding an overlay with low *TCD* and *MRT* can have a significant impact on delay of transferring messages. That is, the total delay of messages which may occur during the operation of the network significantly decrease.
- The overhead of control messages of the *xOverlay* algorithm (*CMO*) is higher than *mOverlay* algorithm. This is because *mOverlay* algorithm does not have any

Table 6 The impact of the underlay topology on the performance of *mOverlay*, *lOverlay* and the *xOverlay* with respect to *CMO*

	Topology		
	Topology T.2	Topology T.3	Topology T.4
lOverlay	112,164 ± 7157	102,106 ± 4911	92,203 ± 5301
mOverlay	91,223 ± 6254	89,187 ± 2521	79,105 ± 4237
xOverlay	102,145 ± 7189	98,055 ± 3936	88,011 ± 5154

adaptation mechanism for overlay structure and no control messages will be generated for this purpose. A higher value for *CMO* throughout the operation of the network is the price that we pay for structure adaptation which leads to lower *TCD* and *MRT*.

- Experimentations have shown that *xOverlay* algorithm improves its performance with respect to *TCD* and *MRT* as the number of rounds increases whereas the improvement obtained by *mOverlay* or *lOverlay* algorithms with respect to *TCD* and *MRT* remains unchanged after a certain number of rounds.

6 Conclusions and future works

In this paper, a new closed asynchronous dynamic model of *CLA* whose structure and the number of *LAs* of each cell may vary with time was introduced. In this model, in contrast to the existing models of *CLA*, the number of *LAs* in each cell may vary with time. To show the potential of this new model in designing algorithms an adaptive landmark clustering algorithm was designed for solving topology mismatch problem in unstructured peer-to-peer networks based on this model. To show the superiority of the proposed landmark clustering algorithm, it is compared with two existing algorithms *mOverlay* and *lOverlay*. The results of experimentations showed that the proposed algorithm performs better than the existing algorithms with respect to communication delay and average round-trip time between peers within clusters. One of the advantages of the proposed algorithm is that it has fewer parameters than *lOverlay* algorithm.

The proposed model of the *CLAs* with such dynamic structure can be used to solve problems that can be modeled as a dynamic graph in which an appropriate cellular structure and a set of states for the cells can be defined. As future work, we plan to find new applications for the proposed model in areas such as cloud computing, grid computing, and internet of things.

Appendix 1

The detailed descriptions of the phases of the proposed algorithm are given in this section. Before we present the proposed algorithm in more details, we need to define the followings.

1. Eleven types of messages (*Hello*, *Alive*, *ClusteringRequest*, *ClusteringReply*, *SetRole*, *WarmUpCell*, *ReadyCell*, *AutomatonActivatonRequest*, *AutomatonActivatonReply*, *ReinforcementSignal*, and *NewConfig*) are used in the proposed algorithm.
2. *Meeting point* (*MP*) is a global cache in the network which provides an entry point to the overlay for the new peers.
3. Each peer uses function *M-Activate* that is shown in Fig. 25 to activate the cell of its corresponding cluster. Since in the proposed algorithm, the learning automata of the cells are distributed among peers of the network, and the design of function *Activate* of *CADCLA-VL* that is shown in Fig. 3 is not appropriate for peer-to-peer networks, we designated the function *M-Activate* that is a customized version of function *Activate*. The descriptions of the function *M-Activate* are given in the next paragraph.

In this function, $peer_i$ sends a *WarmUpCell* messages to the landmark peers of the neighboring clusters of its corresponding cluster. This message activates the cells of the neighboring clusters to compute their *restructuring signals*. After sending the *WarmUpCell* messages, $peer_i$ waits for gathering *ReadyCell* messages. $Peer_i$ saves the information of *ReadyCell* messages and calls *automata trigger* function to find the value of *automata trigger signal*. If the value of *automata trigger signal* is true, $peer_i$ activates its learning automaton to select an action and sends *AutomatonActivationRequest* messages to other peers of its corresponding cluster to activate all learning automata of its corresponding cluster to collectively select their actions. After sending *AutomatonActivationRequest* messages, $peer_i$ waits to gather *AutomatonActivationReply* messages. These messages provide required information for executing the *local rule*. $Peer_i$ uses gathered information and executes the *local rule* for finding the *reinforcement signal* of learning automata. The *reinforcement signal* is used to collectively update all learning automata of the peers of the cluster of $peer_i$. Finally, $peer_i$ find the configuration of peers of its cluster using the *structure updating rule* and send *NewConfig* messages to all peers of its corresponding cluster to activate them to change their connections according to the result of *structure updating rule*. Figure 25 shows the pseudo code of the *M-Activate* function.

Now we describe the algorithm that a $peer_i$ executes. The pseudo code of this algorithm is given in Fig. 26. This algorithm has three phases which are described in the rest of this section.

Construction phase: During the *Construction* phase performed by $peer_i$, the peer tries to find an appropriate cluster. In the *Construction* phase, $peer_i$ uses the *Meeting point* to find clusters of the network. $Peer_i$ randomly selects one of the clusters of the network, sends *Hello* message to the landmark peer of the selected cluster and the landmark peers of clusters adjacent to the selected clusters. Finally, $peer_i$ connects to the landmark peer of the selected cluster, sets its role to *Ordinary*, and goes to the *Organization* phase. If $peer_i$ is unable to find any cluster, it sets its role to *Landmark*, registers its identifier into the *Meeting point* as new landmark peer, and goes to *Maintenance* phase.

Algorithm *M-Activate()*

Notation:

The $cluster_x$ denotes the cluster that the *peer* belongs to at any time.
The *cell* denotes the cell corresponds to the *cluster* $Cluster_x$.

Begin
// preparation phase//
Send *WarmUpCell* messages to the landmark peers of the neighboring clusters of cluster $Cluster_x$;
// these messages activates the neighboring cells of the *cell* to compute their *restructuring signals*. //
Compute the *restructuring signal*;
Wait for a certain duration *THRESHOLD* for gathering *ReadyCell* messages;
Save the information of *ReadyCell* messages into the *peer*;
// *ReadyCell* messages containing *restructuring signals* of the neighboring cells of the *cell*.//
Call the *automaton trigger function* to determine the value of *automaton trigger signal*;
// function *automaton trigger function* uses information saved in the *peer*.//
// structure updating phase//
If (the value of the *restructuring signal* is equal to 1) **Then**
Find the configuration of peers of cluster $Cluster_x$ using the *structure updating rule*;
Send *NewConfig* messages to all peers which must change their connections;
// this message contains the information about new configuration for peers were determined by the *structure updating rule*.//
Call the *automaton trigger function* to determine the value of *automaton trigger signal*;
EndIf
//state updating phase//
If (the value of *automaton trigger signal* is equal to true) **Then**
Activate the learning automaton of the *peer* to select an action;
Send *AutomatonActivationRequest* messages to all peers of the cluster $Cluster_x$;
// these messages activate *learning automata* of the *cell* to choose their actions.//
Wait for a certain duration *THRESHOLD* for gathering *AutomatonActivationReply* messages;
Save the information of *AutomatonActivationReply* messages into the *peer*;
Compute the *reinforcement signal* using the *local rule*; // using information gathered in the peer//
Send *ReinforcementSignal* messages to all peers of the cluster $Cluster_x$;
Update the *learning automaton* of the *peer;*// *reinforcement signal* has been computed by the *local rule*.//
EndIf
End

Fig. 25 Pseudo code of the procedure which a peer executes for using the cell of its corresponding cluster

Organization phase: During the *Organization* phase performed by $peer_i$, $peer_i$ and the peers that they have the same cluster as the cluster of $peer_i$ try to find an appropriate landmark peer for their cluster using the *CADCLA-VL*. In the *Organization* phase, $peer_i$ activates the cell of its corresponding cluster by

Algorithm *xOverlay* ()

Notation:

The *Meeting point* denotes the identifier of a well-known peer which saves the identifiers of the landmark peers of the network.
The $cluster_x$ denotes the cluster that the *peer* belongs to at any time.
The *cell* denotes the cell corresponds to the *cluster* $Cluster_x$.

```
01 Begin
02                                  //Construction phase//
03   Select a random cluster Cluster_x from Meeting point;
04   If (Cluster_x is null) Then
05     Set the role of the peer to Landmark;
06     Register the peer in the Meeting point as new landmark peer;// each landmark
                                                                    peer manages
                                                                    a cluster//
07     Goto Maintenance phase; // goto line 21//
08   Else
09     Connect to the landmark peer of cluster Cluster_x;
10     Send Hello message to the landmark peer of the cluster Cluster_x and the
                  landmark peers of clusters adjacent to the cluster Cluster_x;
11     Set the role of the peer to Ordinary;
12   EndIf
13                                  //Organization phase//
14   Call function M-Activate to activate the cell; // the pseudo code of this function is
                                                       given in Fig. 25//
15   Send ClusteringRequest messages to all peers of cluster Cluster_x;
16   Waits for a certain threshold THRESHOLD for gathering ClusteringReply
                                                                  messages;
17   Save information of ClusteringReply messages into the peer;
18   Select an landmark peer;
     // select one of the peers (including the peer) which its probability of selection of
       "change the role to landmark peer" action is higher than other peers.//
19   Set the role of the peer; // if the peer has been selected as landmark peer the peer
                                 sets its role to Landmark and sets its role to Ordinary,
                                 otherwise.//
20   Send SetRole message to all the peers of the cluster Cluster_x;
      // this message contains the identifier of the selected landmark peer.
21                                  //Maintenance phase//
22   maintainInterrupt ←"False";
23   While (maintainInterrupt ="False") Do
24     If (the role of the peer is Landmark) Then
25       Send Alive message to all peers of cluster Cluster_x;
         // this message contains information about new peers of cluster Cluster_x.//
26     EndIf
27     Wait for a certain duration THRESHOLD until receiving Alive,
        ClusteringRequest, Hello, SetRole, WarmUpCell, AutomatonActivatonRequest
        , and NewConfig messages;
28     If (no message has been received or the connections of the peer has been
         changed) Then
```

Fig. 26 Pseudo code of the proposed algorithm

```
29      If (the role of the peer is Ordinary) Then
30        maintainInterrupt ←"True";
31      EndIf
32    EndIf
33    If (a ClusteringRequest message has been received from peer_j) Then
34      Save the information of ClusteringRequest message into the peer;
35      Send a ClusteringReply message to peer_j;
          //this message contains the action probability of selecting the "change the
          role to landmark peer" action of the learning automaton of the peer.//
36    EndIf
37    If (a Hello message or an Alive message has been received from peer_j) Then
38      Save the information of the received message into the peer;
39      Connect to peer_j;
40    EndIf
41    If (a SetRole message has been received from peer_j) Then
42      Set the identifier of its cluster to the identifier of landmark peer reported by
                                                          the SetRole message;
        // If the peer has been selected as a new landmark peer, then the peer sets its
          role to Landmark, otherwise the peer sets its role to Ordinary.//
43      Send a Hello message to the landmark peer;
44    EndIf
45    If (a WarmUpCell message has been received from peer_j) Then
46      If (the peer_j belongs to neighboring cluster of cluster Cluster_x) Then
          Compute the restructuring signal;
47        Send a ReadyCell message to peer_j;// this message contains the computed
                                                  restructuring signal.//
48      EndIf
49    EndIf
50    If (a AutomatonActivatonRequest message has been received from peer_j) Then
51      Activate the learning automaton of the peer to choose an action;
52      Send a AutomatonActivatonReply message to peer_j;
        // this message contains the action selected by the learning automaton of the
          peer, information about delays between the peer and other peers of the
          cluster Cluster_x, and information about delays between the peer and the
          landmark peers of the clusters adjacent to cluster Cluster_x.//
53      Receive ReinforcementSignal message from peer_j;
54      Update the learning automaton of the peer;
          // the reinforcement signal for updating the learning automaton has been
            reported by the ReinforcementSignal message.//
55    EndIf
56    If (a NewConfig message has been received from peer_j) Then
57      Change the neighbors of the peer using information reported by the
                                                          NewConfig message;
58    EndIf
59  EndWhile
60  Goto Organization phase;//goto line 13//
61  End
```

Fig. 26 continued

executing function *M-Activate()*. Then $peer_i$ sends *ClusteringRequest* messages to all peers of its corresponding cluster to inform them that their landmark peer is not valid. After sending the *ClusteringRequest* message, $peer_i$ waits for a certain threshold *THRESHOLD* and gathers *ClusteringReply* messages which consist of information about learning automata of other peers of the cluster. $peer_i$ selects one of peers (including $peer_i$) which the probability of selection of change the role to landmark peer action of its learning automaton is maximum. Finally, $peer_i$ sets its role and sends *SetRole* messages containing the selected landmark peer to the peers of its corresponding cluster. Note that, if the selected landmark peer is $peer_i$, $peer_i$ sets its role to *Landmark* and sets its role to *Ordinary*, otherwise. At the end of *Organization* phase, $peer_i$ goes to the *Maintenance* phase.

Maintenance phase: In this phase, if the role of $peer_i$ is *Landmark*, $peer_i$ sends *Alive* messages to all the peers of its corresponding cluster. *Alive* message contains information about new peers of the cluster of $peer_i$. $Peer_i$ waits for a certain duration *THRESHOLD* to receive one of the *ClusteringRequest*, *Hello*, *Alive*, *SetRole*, *WarmUpCell*, *AutomatonActivationRequest*, and *NewConfig* messages.

$Peer_i$, upon receiving a *ClusteringRequest* message from a $peer_j$, saves the information of *ClusteringRequest* message. Then, $peer_i$ produces a *ClusteringReply* message containing the action probability of selecting the "change the role to landmark peer" action of its learning automaton. After sending the *ClusteringReply* message to $peer_j$, $peer_i$ restarts the *Maintenance* phase.

$Peer_i$, upon receiving a *Hello* message or an *Alive* message from a $peer_j$, saves information of $peer_j$, connects to $peer_j$, and restarts the *Maintenance* phase.

$Peer_i$, upon receiving a *SetRole* message from a $peer_j$ resided in its corresponding cluster, sets the identifier of its cluster to the identifier of the new landmark peer reported by the *SetRole* message. If $peer_i$ has been selected as a new landmark peer, then $peer_i$ sets its role to *Landmark*, otherwise $peer_i$ sets its role to *Ordinary* and sends a *Hello* message to the new landmark peer. Finally, $peer_i$ restarts the *Maintenance* phase.

$Peer_i$, upon receiving a *WarmUpCell* message from a $peer_j$ which $peer_j$ belongs to neighboring clusters of the cluster of $peer_i$, computes the restructuring signal using the restructuring function and sends a *ReadyCell* message to the $peer_j$. The *ReadyCell* message contains the computed restructuring signal. Finally, $peer_i$ restarts the *Maintenance* phase.

$Peer_i$, upon receiving an *AutomatonActivatonRequest* message from a $peer_j$, activates its learning automaton and sends *AutomatonActivatonReply* message to $peer_j$. This message contains the action selected by the learning automaton of $peer_i$, information about delays between $peer_i$ and other peers of the cluster of $peer_i$, and information about delays between $peer_i$ and the landmark peers of the clusters adjacent to cluster of $peer_i$. Then, $peer_i$ waits until receives a *ReinforcementSignal* message from $peer_j$. After receiving the *ReinforcementSignal* message, $peer_i$ updates its learning automaton using *reinforcement signal* reported by the *ReinforcementSignal* message and restarts the *Maintenance* phase.

$Peer_i$, upon receiving a *NewConfig* message from a $peer_j$, *changes* its connections to its neighbors using information reported by the *NewConfig* message. Then, $peer_i$ restarts the *Maintenance* phase.

If $peer_i$ does not receive any message during the specified period of *THRESHOLD* or the connections of the $peer_i$ are changed, $peer_i$ checks its role. If the role of $peer_i$ is *Ordinary*, $peer_i$ goes to the *organization* phase, and restarts the *Maintenance* phase otherwise.

Appendix 2

Acronyms	Definition
CA	Cellular Automaton
CMO	Control Message Overhead
CLA	Cellular Learning Automaton
CADCLA	Closed Asynchronous Dynamic Cellular Learning Automaton
CADCLA-VL	A Closed Asynchronous Dynamic Cellular Learning Automaton with multiple learning automata in each cell
DCLA	Dynamic Cellular Learning Automaton
DICLA	Dynamic Irregular Cellular Learning Automaton
HDICLA	Heterogeneous Dynamic Irregular Cellular Learning Automaton
hOverlay	A version of the proposed algorithm in which the learning automata of *CADCLA-VL* are deactivated
kOverlay(p)	A version of the proposed algorithm which its *CADCLA-VL* is tuned with an algorithm described as follows. Upon activating a cell of *CADCLA-VL* the information about each peer of its corresponding cluster will be used with probability *p*
LA	Learning Automata
lOverlay	An extension of the *mOverlay* algorithm
MRT	Mean Round-trip Time
M-Activate	A function that executes the cell activation process of the *CLA* in distributed manner
MP	*MP* is a global cache in the network which provides an entry point to the overlay for the new peers
mOverlay	A well-known landmark clustering algorithm
nOverlay	A version of the proposed algorithm in which *CADCLA-VL* is replaced with a pure-chance *CADCLA-VL*
OADCLA	Open Asynchronous Dynamic Cellular Learning Automaton
PE	Potential Energy
SCLA	Static Cellular Learning Automaton
TCD	Total Communication Delay
xOverlay	The proposed *CLA* based algorithm

References

1. S. Wolfram, *A New Kind of Science* (Wolfram Media, Champaign, 2002)
2. M. Cook, Universality in elementary cellular automata. Complex Syst. **15**(1), 1–40 (2004)
3. N. Ganguly, A. Deutsch, A cellular automata model for immune based search algorithm, in *6 th International Conference on Cellular Automata for Research and Industry*, Amsterdam, Netherlands (2004), pp. 142–150

4. J.M. Benito, P. Hernández, *Modelling Segragation Through Cellular Automata: A Theoretical Answer* (Instituto Valenciano de Investigaciones Económicas, València, 2007)
5. M. Esnaashari, M.R. Meybodi, Data aggregation in sensor networks using learning automata. Wirel. Netw. **16**(3), 687–699 (2010)
6. H. Beigy, M.R. Meybodi, A learning automata-based adaptive uniform fractional guard channel algorithm. J. Supercomput. **71**(3), 871–893 (2015)
7. 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. **30**(17), 1-22 (2017)
8. S. Gholami, M.R. Meybodi, A.M. Saghiri, A learning automata-based version of SG-1 protocol for super-peer selection in peer-to-peer networks," in *Proceedings of the 10th International Conference on Computing and Information Technology*, Angsana Laguna, Phuket, Thailand (2014), pp. 189–201
9. M. Ghorbani, A. Saghiri, M. Meybodi, A novel learning based search algorithm for unstructured peer to peer networks. Tech. J. Eng. Appl. Sci. **3**(2), 145–149 (2013)
10. M. Ghorbani, M.R. Meybodi, A.M. Saghiri, A novel self-adaptive search algorithm for unstructured peer-to-peer networks utilizing learning automata, in *3rd Joint Conference of AI & Robotics and 5th RoboCup Iran Open International Symposium*, Qazvin, Iran (2013), pp. 1–6
11. A.M. Saghiri, M.R. Meybodi, A self-adaptive algorithm for topology matching in unstructured peer-to-peer networks. J. Netw. Syst. Manag. **24**(2), 393-426 (2016)
12. H. Beigy, M.R. Meybodi, A mathematical framework for cellular learning automata. Adv. Complex Syst. **3**(4), 295–319 (2004)
13. 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*, Tehran, Iran (2007), pp. 21–28
14. M. Esnaashari, M. Meybodi, A cellular learning automata-based deployment strategy for mobile wireless sensor networks. J. Parallel Distrib. Comput. **71**(5), 988–1001 (2011)
15. M. Esnaashari, M. Meybodi, Deployment of a mobile wireless sensor network with k-coverage constraint: a cellular learning automata approach. Wirel. Netw. **19**(5), 945–968 (2013)
16. 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**, 1216–1226 (2015)
17. R. Rastegar, M.R. Meybodi, A new evolutionary computing model based on cellular learning automata, in *IEEE Conference on Cybernetics and Intelligent Systems*, vol. 1, Singapore (2004), pp. 433–438
18. R. Rastegar, M.R. Meybodi, A. Hariri, A new fine-grained evolutionary algorithm based on cellular learning automata. Int. J. Hybrid Intell. Syst. **3**(2), 83–98 (2006)
19. M. Mozafari, M.E. Shiri, H. Beigy, A cooperative learning method based on cellular learning automata and its application in optimization problems. J. Computat. Sci. **11**, 279–288 (2015)
20. H. Beigy, M.R. Meybodi, Open synchronous cellular learning automata. Adv. Complex Syst. **10**(4), 527–556 (2007)
21. H. Beigy, M.R. Meybodi, Asynchronous cellular learning automata. Automatica **44**(5), 1350–1357 (2008)
22. 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 Cybern. **40**(1), 54–65 (2010)
23. M. Esnaashari, M.R. Meybodi, Irregular cellular learning automata. IEEE Trans. Cybern. **99**, 1 (2014)
24. A.M. Saghiri, M.R. Meybodi, An approach for designing cognitive engines in cognitive peer-to-peer networks. J. Netw. Comput. Appl. **70**, 17–40 (2016)
25. X.Y. Zhang, Q. Zhang, Z. Zhang, G. Song, W. Zhu, A construction of locality-aware overlay network: mOverlay and its performance. IEEE J. Sel. Areas Commun. **22**(1), 18–28 (2004)
26. T. Qiu, E. Chan, M. Ye, G. Chen, B.Y. Zhao, Peer-exchange schemes to handle mismatch in peer-to-peer systems. J. Supercomput. **48**(1), 15–42 (2009)
27. H.-J. Ju, L.-J. Du, Nodes clustering method in large-scale network, in *8th International Conference on Wireless Communications, Networking and Mobile Computing*, Shanghai, China (2012), pp. 1–4
28. Y. Liu, A two-hop solution to solving topology mismatch. IEEE Trans. Parallel Distrib. Syst. **19**, 1591–1600 (2008)
29. H.C. Hsiao, H. Liao, P.S. Yeh, A near-optimal algorithm attacking the topology mismatch problem in unstructured peer-to-peer networks. IEEE Trans. Parallel Distrib. Syst. **21**(7), 983–997 (2010)

30. H.C. Hsiao, H. Liao, C.C. Huang, Resolving the topology mismatch problem in unstructured peer-to-peer networks. IEEE Trans. Parallel Distrib. Syst. **20**(11), 1668–1681 (2009)
31. Y. Li, Z. Yu, An improved genetic algorithm for network nodes clustering, in *Proceedings of the Second International Conference on Information Computing and Applications*, Qinhuangdao, China (2011), pp. 399–406
32. Y. Jiang, J. You, X. He, A particle swarm based network hosts clustering algorithm for peer-to-peer networks, in *International Conference on Computational Intelligence and Security*, vol. 2, Guangzhou, China (2006), pp. 1176–1179
33. R. Tian, Y. Xiong, Q. Zhang, B. Li, B.Y. Zhao, X. Li, Hybrid overlay structure based on random walks, in *Peer-to-Peer Systems IV* (Springer, 2005), pp. 152–162
34. M. Scheidegger, T. Braun, Improved locality-aware grouping in overlay networks, in *Kommunikation in Verteilten Systemen*, Bern, Switzerland (2007), pp. 27–38
35. S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "Topologically-aware overlay construction and server selection," in *The 21st Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 3, New York, USA (2002), pp. 1190–1199
36. S. Wolf, and P. Merz, "Evolutionary local search for the super-peer selection problem and the p-hub median problem," in *Proceedings of the 4th International Conference on Hybrid Metaheuristics*, Berlin, Heidelberg (2007), pp. 1–15
37. Z. Xu, C. Tang, Z. Zhang, Building topology-aware overlays using global soft-state, in *23rd International Conference on Distributed Computing Systems*, Providence, RI, USA (2003), pp. 500–508
38. K.S. Narendra, M.A.L. Thathachar, *Learning Automata: An Introduction* (Prentice-Hall, Englewood Cliffs, 1989)
39. M. Thathachar, P.S. Sastry, *Networks of Learning Automata: Techniques for Online Stochastic Optimization* (Kluwer Academic Publishers, Dordrecht, Netherlands, 2004)
40. M.E. O'kelly, A quadratic integer program for the location of interacting hub facilities. Eur. J. Oper. Res. **32**(3), 393–404 (1987)
41. F. Aurenhammer, Voronoi diagrams—a survey of a fundamental geometric data structure. ACM Comput. Surv. (CSUR) **23**(3), 345–405 (1991)
42. B.A. Bash, P.J. Desnoyers, Exact distributed Voronoi cell computation in sensor networks, in *Proceedings of the 6th International conference on Information Processing in Sensor Networks*, New York, NY, USA (2007), pp. 236–243
43. W.-P. Chen, J.C. Hou, L. Sha, Dynamic clustering for acoustic target tracking in wireless sensor networks. IEEE Trans. Mob. Comput. **3**(3), 258–271 (2004)
44. M. Sharifzadeh, C Shahabi, Supporting spatial aggregation in sensor network databases, in *Proceedings of the 12th Annual ACM International Workshop on Geographic Information Systems*, Washington DC, USA (2004), pp. 166–175
45. Z. Zhou, S.R. Das, H. Gupta, Variable radii connected sensor cover in sensor networks. ACM Trans. Sens. Netw. (TOSN) **5**(1), 8 (2009)
46. I. Baumgart, B. Heep, S. Krause, OverSim: a scalable and flexible overlay framework for simulation and real network applications," in *Peer-to-Peer Computing*, Seattle, Washington, USA (2009), pp. 87–88
47. L. Li, D. Alderson, W. Willinger, J. Doyle, A first-principles approach to understanding the internet's router-level topology. ACM SIGCOMM Comput. Commun. Rev. **34**(4), 3–14 (2004)
48. P. Mahadevan, D. Krioukov, M. Fomenkov, B. Huffaker, X. Dimitropoulos, A. Vahdat, *Lessons from Three Views of the Internet Topology* (University of California, San Diego, 2005). **(tr-2005-02)**
49. B. Huffaker, D. Plummer, D. Moore, K.C. Claffy, Topology discovery by active probing, in *Proceedings of Symposium on Applications and the Internet Workshops*, Washington, DC, USA (2002), pp. 90–96