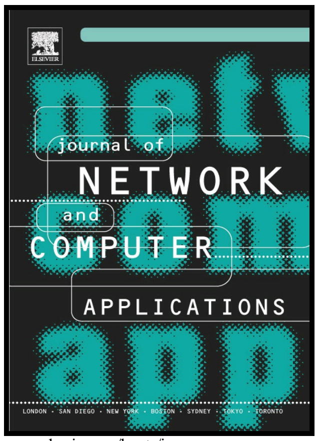
journal of
NETWORK
and
COMPUTER
APPLICATIONS
LONDON • SAN DIEGO • NEW YORK • BOSTON • SYDNEY • TOKYO • TORONTO

www.elsevier.com/locate/jnca

An Approach for Designing Cognitive Engines in Cognitive Peer-to-Peer Networks

Ali Mohammad Saghiri, Mohammad Reza Meybodi

PII: S1084-8045(16)30108-4
DOI: http://dx.doi.org/10.1016/j.jnca.2016.05.012
Reference: YJNCA1654

To appear in: *Journal of Network and Computer Applications*

Received date: 28 September 2015
Revised date: 24 February 2016
Accepted date: 10 May 2016

An Approach for Designing Cognitive Engines in Cognitive Peer-to-Peer Networks

Ali Mohammad Saghiri[1], Mohammad Reza Meybodi[2]

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

Telephone number: +98-21-64545120

E-mail Address: a_m_saghiri@aut.ac.ir

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

Telephone number: +98-21-64545120

E-mail Address: mmeybodi@aut.ac.ir

Corresponding Author:

Ali Mohammad Saghiri

An Approach for Designing Cognitive Engines in Cognitive Peer-to-Peer Networks

Ali Mohammad Saghiri
Soft Computing Laboratory
Amirkabir University of Technology
Tehran, Iran
a_m_saghiri@aut.ac.ir

Mohammad Reza Meybodi
Soft Computing Laboratory
Amirkabir University of Technology
Tehran, Iran
mmeybodi@aut.ac.ir

Abstract-A cognitive network is a network which can learn to improve its performance while operating under its unknown and dynamic environment. Cognitive engine as part of a cognitive network tries to adaptively find an appropriate configuration for the network. Up until now no peer-to-peer network management algorithm has been designated utilizing cognitive networking concepts. In this paper, we adopt cognitive networking concepts and present a framework for cognitive peer-to-peer networks and then propose an approach based on cellular learning automata for designing cognitive engines for solving network management problems in peer-to-peer networks. To show the potential of the proposed approach, a cognitive engine for solving topology mismatch problem in unstructured peer-to-peer networks will be presented. To evaluate the proposed approach, computer simulations have been conducted using the cognitive engine designed for solving topology mismatch problem and then the results are compared with the results obtained for two existing algorithms called *PROP-O* and *X-BOT* for solving topology mismatch problem. It has been shown that the proposed cognitive engine performs better than the existing algorithms with respect to end-to-end delays and delays of mismatched paths.

Keywords: Cognitive Network; Cognitive Engine; Cellular Learning Automata; Peer-to-Peer Network; Topology Matching.

1 Introduction

A cognitive network learns from the past experiences and improves its decisions about its configuration. Cognitive networks are defined as networks with cognitive processes capable of learning from the results of their actions. A cognitive process recognizes current network situations (plans, conditions, etc) and acts based on them [1]. The cognitive networks such as cognitive radio networks [2]–[5], cognitive sensor networks [6], cognitive wireless mesh networks [7], cognitive mobile ad-hoc networks [8], and cognitive personal networks [9] have received much attention in recent years. To our knowledge no network management algorithm based on cognitive networking concept for peer-to-peer network has been reported in the literature. In [10]–[12], a general framework for cognitive networks based on end-to-end goals and cognitive processes has been reported by Thomas et al. This framework determines the functionality of different elements of a cognitive network. In this framework, the cognitive engine which consists of several cognitive processes is responsible for managing the network configurations. A detailed description of this framework is given later in subsection 2.5.

Different artificial intelligence techniques are used to design cognitive engines in different types of cognitive networks. Random Neural Networks in cognitive packet networks [8], [13], [14], Q-Learning in cognitive radio networks [6], [15], learning automata in cognitive mesh networks [7], [16], ant colony in cognitive radio networks [17], [18], and genetic algorithm in cognitive radio networks [19], [20] are used to design cognitive engines.

Since peer-to-peer networks are large and also dynamic, designing management algorithms for them is a difficult problem[21]. In recent years, this problem has become more challenging because the peer-to-peer networks have been merged into other systems such as cloud computing, content distribution networks, and social networks. It should be note that, in a hybrid system such as peer-to-peer cloud [22], [23], peer-to-peer content distribution network[24]–[26], and peer-to-peer social networks[27], [28] the number of problems in designing the management algorithms of peer-to-peer network will be increased. Therefore a new approach such as cognitive networking for solving the management problems in peer-to-peer networks can be useful. The cognitive networking concept has not been introduced in the domain of peer-to-peer networks, but some similarities between management algorithms of peer-to-peer networks and cognitive networks are exist which some of them are described in the next three paragraphs.

Several artificial intelligence techniques are used to design management algorithms in different types of peer-to-peer networks. Among artificial intelligence techniques self-organized algorithms are widely used in peer-to-peer networks. Ant colony algorithm is used to implement a resource management algorithm in unstructured peer-to-peer networks [29], [30]. In [31], the structure of super-peers in super-peer based peer-to-peer networks is managed by a model inspired form growth pattern of fungi. In [32], a management algorithm based on growing neural gas model for super-peer management in super-peer based peer-to-peer networks is reported. A search algorithm for unstructured peer-to-peer networks based on cellular automata [33] and a search algorithm inspired from bacterial foraging strategy for hierarchical unstructured peer-to-peer networks [34] are also reported in the literature. In [35], Schelling segregation model which is a self-organizing model in ecological science is used for designing a topology clustering algorithm in unstructured peer-to-peer networks. It should be noted that, artificial intelligence techniques are employed in the cognitive processes of the cognitive networks and then the cognitive processes are used to analyze the functionality of the cognitive network. Since the existing artificial intelligence based peer-to-peer networks are not designated based on cognitive networking frameworks, they cannot be considered as cognitive networks.

Several algorithms based on measurement-driven approach not in cognitive networking framework are used in designing management algorithms in peer-to-peer networks. In this approach, we find properties of peers using extensive measurements and then design algorithms that exploit those properties in order to solve management problems of peer-to-peer networks [36]–[39]. In [36], [38], a topology adaption algorithm based on measurement driven approach is reported. In this algorithm, peers organize themselves into an interest-based structure on top of a peer-to-peer network. This algorithm uses the relationship among peers and contents and then changes the links of the peers in order to decrease the traffic of flooding techniques in the network. This algorithm is also used in peer-to-peer social networks reported in[27]. It should be noted that, measurement-driven approach is implicitly used for topology adaption algorithms reported in [39]–[42].

A main drawback of the existing artificial intelligence based management algorithms and measurement-driven based algorithms is that they are not designated based on a general and uniform framework. Therefore we cannot simply analyze and use them. Since cognitive networking concept has shown its potential in improving the design of management algorithms in different types of computer networks such as sensor networks and radio networks, we may use it to mitigate the problems of designing management algorithms in peer-to-peer networks[1].

In this paper, we adopt the framework of cognitive networks reported by Thomas et al. in [10], [43] for peer-to-peer networks and propose an approach based on cellular learning automata for designing cognitive engines for solving management problems in peer-to-peer networks. To our knowledge no management mechanism based on cellular learning automata has been reported for either peer-to-peer networks or cognitive networks in the literature. To show the potential of the proposed approach, a cognitive engine for solving topology mismatch problem in unstructured peer-to-peer networks will be presented. The ideas according to which the rule of cellular learning automata are obtained are borrowed from PROP-O algorithm [40] and Schelling segregation model [44], [45]. To evaluate the proposed approach for designing cognitive engines, computer simulations have been conducted and then the results are compared with the results obtained for two existing algorithms called PROP-O[40] and X-BOT[46] for solving topology mismatch problem. It has been shown that the proposed cognitive engine produces results which are superior to existing algorithms with respect to end-to-end delays and delays of mismatched paths.

The rest of this paper is organized as follows. Section 2 is dedicated to some preliminaries used in this paper. Section 3 presents a framework for cognitive peer-to-peer networks. In section 4, an approach based on cellular learning automata for designing cognitive engines in peer-to-peer networks and its application to solve topology mismatch problem is proposed. Section 5 reports the results of experimentations and section 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 cellular learning automata, topology mismatch problem, PROP-O algorithm, Schelling segregation model, and a framework introduced by Thomas et al. for cognitive networks.

2.1 CELLULAR LEARNING AUTOMATA

In this section, cellular automata, learning automata, and cellular learning automata are briefly reviewed.

2.1.1 Cellular Automata

Cellular Automata (*CAs*) are composed of independent and identical cells that arranged into a fixed lattice. In *CA*, each cell can select a state from a finite set of states. The cells update their states according to a local rule. The new state of each cell depends on the previous states of a set of cells, including the cell itself, and its neighbors. The updating of all cells can be made in synchronous or asynchronous fashion. It evolves in discrete time steps [47]. In some applications, we need new models of *CA* to cover different form of dynamicity in the model. Different types of dynamicity are analyzed in domain of *CA* which some of them are reviewed as follow. In [48], a model of cellular automaton is proposed that the state transition rules can be selected dynamically. In [49] and [50] two models of cellular automata are proposed that the connections between the cells are allowed to be changed according to rules similar to the state transition rules associated with the traditional *CA*. This means that, links between cells may be created or destroyed during cellular automata evolution.

2.1.2 Learning automata

Learning Automata (*LAs*) are adaptive decision-making devices that operate in random environments. A learning automaton has a finite set of actions and each action has a certain probability (unknown for the automaton) for getting rewarded by its environment. The aim is to learn to choose the optimal action (i.e. the action with the highest probability of being rewarded) through repeated interaction with the environment. If the learning algorithm is chosen properly, then the iterative process of interacting on the environment can be made to result in selection of the optimal action. The interaction between learning automaton and the environment is shown in Fig. 1.

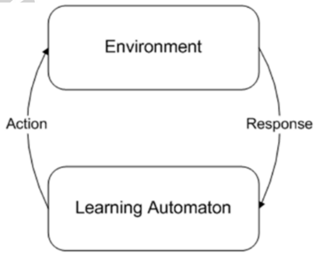

Fig. 1. Learning Automaton (LA)

Learning automata can be classified into two main families, fixed and variable structure learning automata [51], [52]. Variable structure learning automata which is used in this paper is represented by sextuple $\langle\beta, \phi, \alpha, P, G, T\rangle$, where β is a set of inputs actions (called response or reinforcement signal), ϕ is a set of internal states, α a set of outputs, P denotes the state probability vector governing the choice of the state at each stage k, G is the output mapping, and T is learning algorithm. The learning algorithm is a recurrence relation and is used to modify the state probability vector. It is evident that the crucial factor affecting the performance of the variable structure learning automata is learning algorithm for updating the action probabilities. Let α_i be the action chosen at time k as a sample realization from distribution p(k). The linear reward-penalty algorithm (L_{RP}) is one of the earliest schemes. In an L_{RP} scheme the recurrence equation for updating probability vector p is defined by (1) for a favorable response (β=1), and (2) for an unfavorable response (β=0).

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

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

The parameters *a* and *b* represent reward and penalty parameters, respectively. The parameter *a* (*b*) determines the amount of increase (decrease) of the action probabilities.

2.1.3 Cellular Learning Automata

Cellular Learning Automaton (*CLA*) is obtained from combination of *CA* and *LA* [53]. This model combines self-organization concepts from *CA* and learning in unknown environment from *LA*. In *CLA*, a learning automaton is assigned to each cell of cellular automaton. Each cell uses its learning automaton to select its state. In *CLA*, each cell utilizes a local rule to generate the response of its learning automaton to its environment which is the cell itself and its neighbors. Let $cell_j$,$cell_k$, $cell_l$, and $cell_m$ are neighbors of $cell_i$. Fig. 2 shows the interactions of $cell_i$ with its neighbors in a grid like structure. In this figure, the network of learning automata which is used to determine the states of cells $cell_i$, $cell_j$, $cell_k$, $cell_l$, and $cell_m$ is illustrated. Note that, the cellular structures of new models of *CLAs* are not limited to a specific structure. Fig. 3 shows the interaction of learning automaton LA_i of $cell_i$ with its environment. The environment of each learning automaton LA_i residing in cell $cell_i$ is composed of the environment of $cell_i$ and the cells which are in the neighborhood of $cell_i$. The response used by automaton LA_i to update its action probabilities is computed using the local rule. In *CLA*, each cell tries to maximize its expected reward. The expectation of total reward received by all learning automata of *CLA* will be maximized when the pattern of all states of *CLA* converges to a specific pattern which is appropriate for the application which uses *CLA* in its management unit. The local rule creates the relation between the goal of the management unit and the learning process of *CLA*.

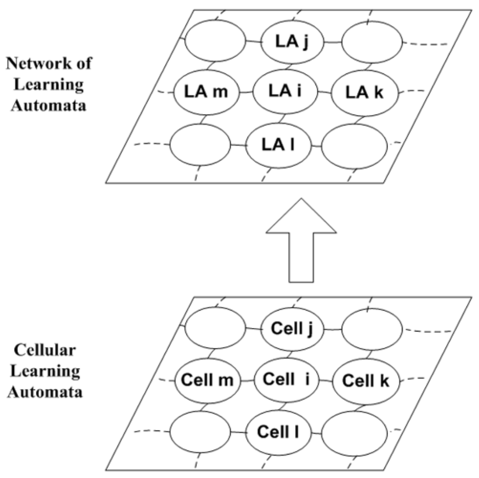

Fig. 2. Cellular Learning Automata

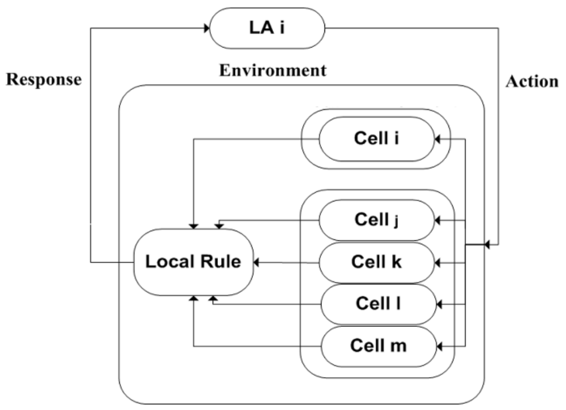

Fig. 3. A Learning Automaton *LA*$_i$ and its local and global environment

CLA can be either synchronous or asynchronous. In synchronous mechanism, all cells use their local rules at the same time [54]. This mechanism implies that there is an external clock which triggers synchronous events. A *CLA* is called asynchronous, if at a given time only some cells are activated independently from each other [55]. During this activation, the state of the rest of the system is held constant. *CLA* can be either closed or open. In closed *CLA*, the action selected by each *LA* depends on the state of its local environment whereas in open *CLA* [54], the action selected by each *LA* depends on both its local environment and an external environment. *CLA* can be either regular or irregular. In standard model of *CLA* [53], the structure of the cells is limited to a lattice. Irregular Cellular Learning Automata (*ICLA*), whose structure can be an arbitrary graph, is suggested for problems that cannot be modeled using a lattice [56]. In [57], a type of *CLA* which is called *CLA* with multiple *LAs* in each has been reported. In some applications, we need to dynamic models of *CLAs* in which the cellular structures are able to dynamically change themselves during computation. Dynamic models of *CLAs* have been recently reported in [58] and [59]. The exiting dynamic modes of *CLAs* support specific forms of dynamicity and we cannot easily extend them for new applications. *CLAs* have found applications in areas such as computer networks [57]–[61], social networks [62], and evolutionary computing [63].

2.2 Topology mismatch problem

Peer-to-Peer networks are overlay networks that are constructed over underlay networks. These networks can be structured or unstructured. In these networks, peers choose their neighbors without considering underlay positions, and therefore the resultant overlay network may have large number of mismatched paths. In a mismatched path a message may meet an underlay position several times which causes redundant network traffic and end-to-end delay[64]. In some of the topology matching algorithms such as *PROP-O* [40], *THANCS* [41], *X-BOT* [46], and one reported in [42], each peer uses a local search operator for gathering information about the neighbors of that peer located in its neighborhood radius. In these algorithms, each peer also uses a local operator for changing the connections among the peers. These matching algorithms reconfigure the overlay structure (using the local operators) in an online fashion in order to solve the topology mismatch problem. These algorithms reconfigure a given overlay graph $G = (V, E)$ to another graph $G^o = (V, E^o)$ by solving the following problem.

$$\min z = \sum_{v\in V} \sum_{u\in V-\{v\}} x_{uv} d_{uv} \quad (3)$$

$$\text{s.t. } \sum_{v\in V} \sum_{u\in V-\{v\}} x_{uv} = 2 \times |E| \quad (4)$$

$$\forall_{U\subset V, U\neq\emptyset} \sum_{uv\in l(U,V)} x_{vu} \geq 1 \quad (5)$$

$$x_{uv} \in \{0,1\} \quad (6)$$

In (3), x_{uv} indicates the existence or absence of a connection between *peer$_u$* and *peer$_v$*. If $x_{uv} = 1$ then there is a connection between *peer$_u$* and *peer$_v$*.d_{uv} is the end-to-end delay from *peer$_u$* and *peer$_v$*. Constraint (4) means that the matching algorithm must not change the number of links in the overlay. In Constraint (5), $l(U,V) = \{vu | v \in U, u \in V - U\}$ is the set of overlay links incident to the peers in the subset $U \subset V$. Constraint (5) indicates that for any subset $U \subset V$ there is at least one overlay link connecting the two components U and $V - U$. Constraint (5) guarantees the connectivity in G. The objective of matching algorithms is to minimize z subject to constraint (4) and (5).

2.3 PROP-O algorithm

PROP-O algorithm is a topology matching algorithm which is able to solve topology mismatch problem in both structured and unstructured peer-to-peer networks [40]. In this algorithm, to solve the topology mismatch problem, a local operator called exchange operator is presented. Fig. 4 shows how the exchange operator exchange equal number of neighbors between $peer_i$ and $peer_j$ if $d_{ki} + d_{lj} > d_{kj} + d_{li}$. In this operator, the edges $(peer_i, peer_k)$ and $(peer_j, peer_l)$ will be replaced by $\left(peer_j, peer_k\right)$ and $(peer_i, peer_l)$ only if $peer_i$ and $peer_j$ are adjacent to each other. With variable neighborhood radius, the exchange operator can be extended if there is a path between $peer_i$ and $peer_j$. In this paper, $\{peer_i, peer_j\}$ and $\{peer_k, peer_l\}$ are called *corresponding peers* and *candidate peers*, respectively. The execution of exchange operator for decreasing the delays of paths of the overlay network leads to decreasing the mismatched paths of the overlay network. The detailed descriptions about this algorithm are given in the rest of this part.

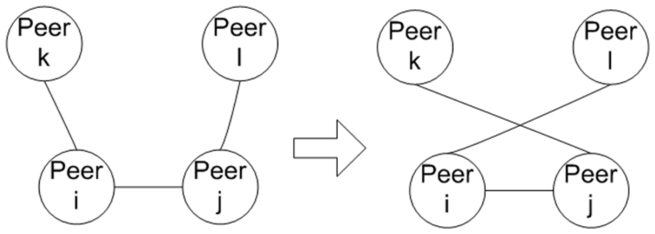

Fig. 4. Exchange Operator

In *PROP-O* algorithm, the process executed by each peer when joining to the network has two major phases: 1. warm-up phase, and 2. maintenance phase. Each phase of this algorithm has two main sub-phases: local search, and exchange. When a new peer *peer*$_i$ joins to the network starts the warm-up phase.

During warm-up phase, *peer*$_i$ gathers information about delays to its neighbors and then starts its local search. The information about neighbors is stored in a priority queue called *neighborQ*. During the local search, *peer*$_i$ selects one of its neighbors such as *peer*$_s$ which is used to find *corresponding* peers then contact to a random *peer*$_v$ as *corresponding* peer which is in a neighborhood radius (variable r_i determines the neighborhood radius) to *peer*$_i$. Then *peer*$_i$ selects some of its neighbors as *candidate* peers and calculates the difference between end-to-end delays before and after exchange operation with *peer*$_v$. The difference between end-to-end delays is stored in a variable called *Var*. After computing the value of variable *Var*, if the exchange operation can decrease the end-to-end delays of links of *peer*$_i$ and *peer*$_v$, then *peer*$_i$ starts its exchange phase. During the exchange phase, *peer*$_i$ utilizes the exchange operator to exchange its *candidate* peers with the *candidate* peers of *peer*$_v$. At the end of warm-up phase, *peer*$_i$ decides to repeat the warm-up phase or goes to maintenance phase. The warm-up phase will be repeated for *MAX_INIT_TRIAL* times which the value of parameter *MAX_INIT_TRIAL* was set by a designer. After warm-up phase, *peer*$_i$ goes to the maintenance phase.

Maintenance phase starts with local search. The local search of the maintenance phase is similar to the local search of warm-up phase. During local search, *peer*$_i$ gathers some information about its neighbors and then contact to a random *peer*$_v$ which is in a neighborhood radius (determined by variable r_i). After computing the value of variable *Var* the peer decide to apply exchange operator or not. If the value of variable *Var* is greater than threshold *MIN_VAR* then *peer*$_i$ change its neighbors using exchange operator. Otherwise the priority of the selected neighbor *peer*$_s$ will be decreased. If an exchange occurs, *peer*$_i$ will decrease the priority number of *peer*$_s$ by a small number like 1 so that *peer*$_s$ could be chosen in near future. Otherwise, *peer*$_s$ will be replaced at the tail of *neighborQ*. After updating the *neighborQ*, *peer*$_i$ decides to repeat the maintenance phase or not using a Timer. In the maintenance phase, the optimization process (consist of local search and exchange) will be repeated till the end of a duration computed based on the result of exchange operation and two parameters *MAX_TIMER* and *INIT_TIMER*. In the maintenance phase, each peer modifies its *neighborQ* if it receives join or leave messages or when its connections changes. In *PROP-O*, a threshold called *m* is defined to determine the numbers of *candidate* peers which can be used in each exchange operation. For more information about *PROP-O* algorithm we refer to [40].

The *PROP-O* algorithm suffers from two problems which are described as follows. The first problem is that there is no adaptive mechanism for setting the neighborhood radius parameter. Finding an appropriate value for this parameter manually is a time consuming process and also error prone. Large neighborhood radius speeds up the convergence of the matching algorithm (because the number of *candidate* peers at each step increases) and it decreases the number of exchanges that must be endured until the convergence of the algorithm. Also, large neighborhood radius causes higher traffic and computational overhead of the network. Small neighborhood radius decreases the number of *candidate* peers at each step of the algorithm which causes the total number of exchanges to be increased. Small neighborhood radius results in lower traffic and computational overhead. Because of the dynamicity of peer-to-peer networks, the operational environment and the neighbors of each peer may change over time (peers continually join and leave the network) and for this reason using a fixed neighborhood radius may not be appropriate. The second problem is the lack of an adaptive mechanism for managing the execution of the exchange operator. Non-adaptive mechanism for managing the execution of the exchange operator leads to performing unnecessary exchange operations which results in increasing the overhead of the matching algorithm (higher number of peers to be reconfigured and extra control messages).

2.4 Schelling segregation model

Schelling segregation model [44], [45] is composed of independent and identical agents. Each agent cares only about the composition of its own local neighborhood. Each agent using a function (called similarity function) calculates the portion of its neighbors which have similar attribute with it. According to a rule called happiness rule each agent decides whether or not to change its neighbors. If the value of similarity function is lower than a parameter z, the agent is unhappy and prefers to change its neighbors in order to increase the number similar neighbors. This process continues until no agent wants to change its neighbors any longer. The happiness rule controls the process of changing the neighborhood in the model.

2.5 A Framework of Cognitive Networks

In this section, we briefly describe the framework introduced by Thomas et al. in [10] for cognitive networks. This framework consists of three layers (Fig. 5): *Requirement Layer*, *Cognitive Process Layer*, and *Software Adaptable Network Layer* as described below.

- ***Requirement Layer*****:** In this layer, the goals and the behavior of the network will be described by a language called *Cognitive Specification Language*.
- ***Cognitive Process Layer*****:** In this layer, cognitive processes observe the changes in the networks using network status sensors of *Software Adaptable Network Layer* and then execute management operators on modifiable elements of *Software Adaptable Network Layer* to change the configuration of nodes in order to find an appropriate configuration for the network. In this layer several cognitive elements are defined. Each cognitive element has an objective and continually observes its environment, analyzes its situation, and finally makes appropriate decision considering its objective. The objectives of cognitive elements are determined in the *Requirement Layer*. In the *Cognitive Process Layer*, a cognitive engine which consists of several cognitive processes is responsible for managing the cognitive elements.
- ***Software Adaptable Network Layer (SAN Layer)*****:** This layer provides the action space for the cognitive processes. This layer consists of the application programming interface, modifiable elements, and network status sensors. A modifiable element is an element which contains a variable used in the management algorithm of the network with possible actions for changing it. Modifiable elements act as points of control in cognitive networks. A network status sensor monitors the changes in the network. Network status sensors are able to execute appropriate functions to find local information (such as node capacity, or lifetime) or global information (such as end-to-end delay) about the network. The status of the network is the source of valuable information for cognitive processes. Modifiable elements and network status sensors should have public interfaces to the application programming interface.

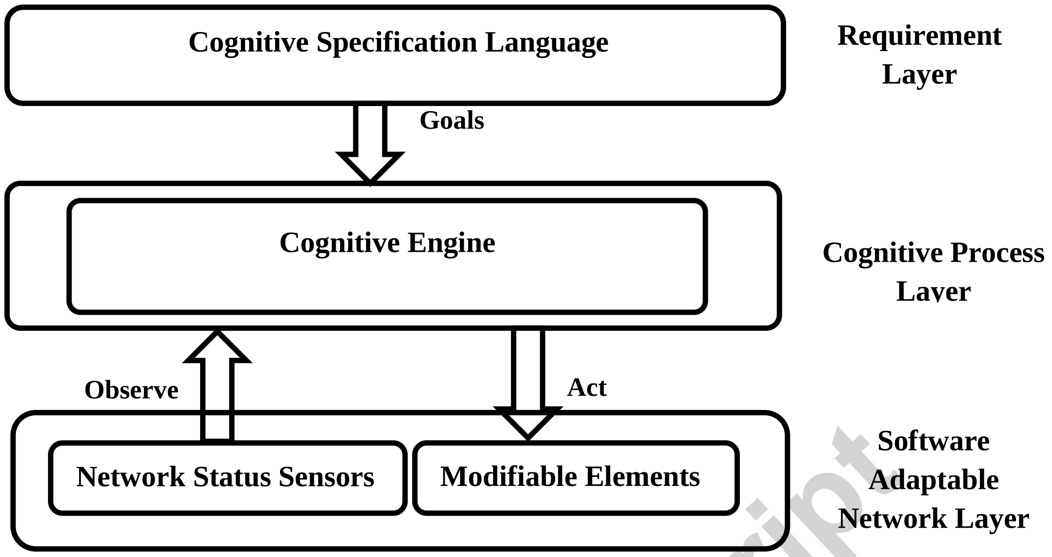

Fig. 5. A framework for cognitive networks reported in [10]

3 A Framework for Cognitive Peer-to-Peer Networks

In this section, we adopt the framework of the cognitive networks introduced by Thomas et al. [10], [43] and present a framework for cognitive peer-to-peer networks. The cognitive peer-to-peer network consists of a set of cognitive peers. Each cognitive peer in the cognitive peer-to-peer network corresponds to a peer in the peer-to-peer overlay network. The topological structure of the cognitive peer-to-peer network is isomorphic to the topological structure of the peer-to-peer overlay network (Fig. 6). The structure of the framework that is used in a cognitive peer is shown in Fig. 7. Each cognitive peer uses the framework of the cognitive networks reported by Thomas et al. As it was previously mentioned, this framework consists of three layers: *Requirement Layer, Cognitive Process Layer* and *SAN Layer*. The definitions of the layers in the presented framework are given in the next three subsections.

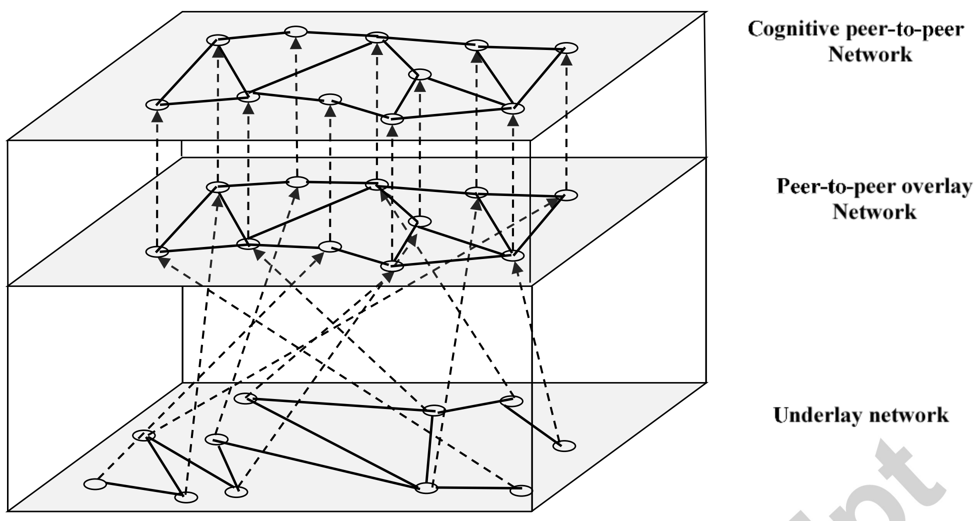

Fig. 6. Cognitive peer-to-peer network

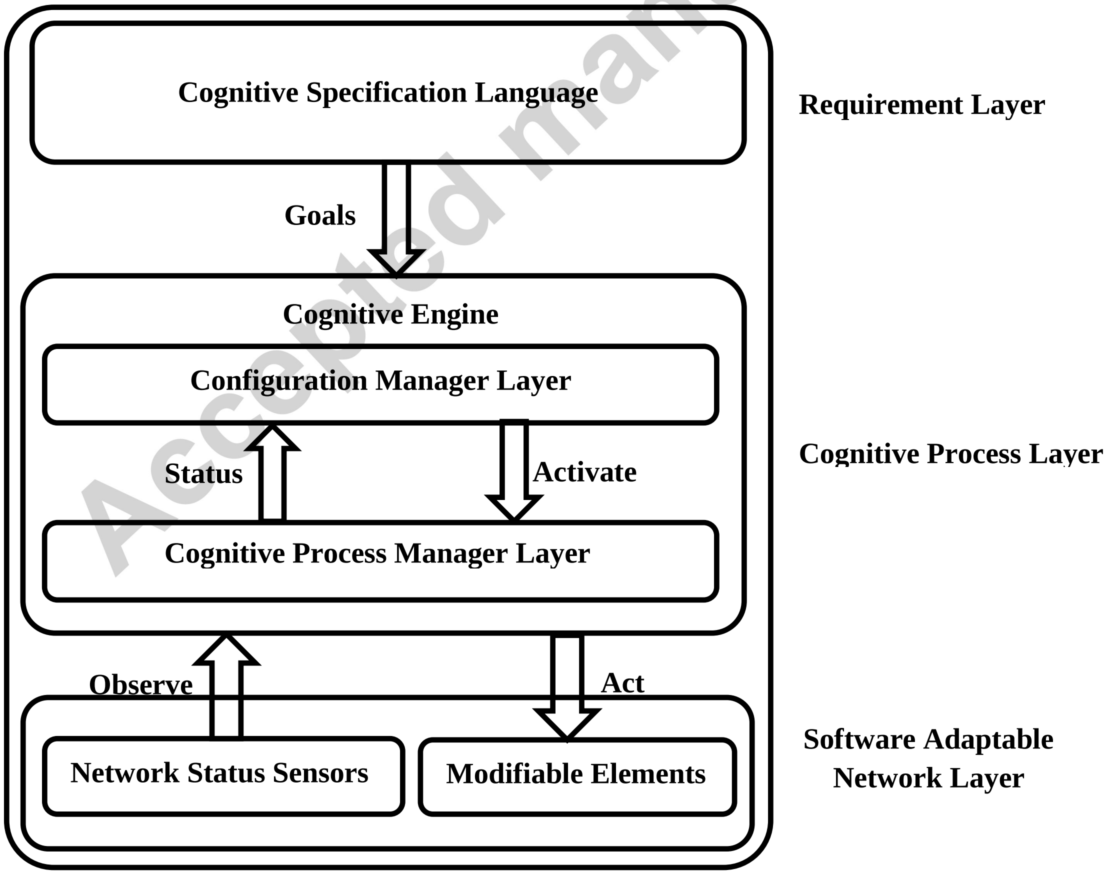

Fig. 7. The framework which is used in a cognitive peer

3.1 Requirement Layer

In the *Requirement Layer*, the goals and the behavior of the network are described by *Cognitive Specification Language*. Each cognitive peer finds a file celled *configuration* file that is shared in the peer-to-peer overlay network. In the *configuration* file, the *Cognitive Specification Language* is used to determine the goals of the cognitive network. Each cognitive peer finds the goals of the cognitive network from the *configuration* file and then transfers them to the *Cognitive Process Layer.* This layer enables the manager to change the goals of the cognitive peer-to-peer network by sharing a new file as *configuration* file. It should be noted that, changing the goals of the cognitive network in the *Requirement Layer* leads to changing the optimizing functions in the *Cognitive Process Layer*.

Several approach considering distributed nature of peer-to-peer network for sharing the *configuration* file are suggested below.

- **Centralized approach:** In this approach, the *configuration* file will be stored in a global well-known peer called *configuration container* peer. Each peer connects to the *configuration container* peer and then finds the last version of the *configuration* file.
- **Semi-centralized approach:** In this approach, the *configuration* file will be stored in some peers called *configuration container* peers in the peer-to-peer overlay network. The *configuration container* peers can be selected in static or dynamic manner. In super-peer based overlay networks, the super-peers can be used as *configuration container* peers. *Configuration container* peers are in charge of updating the *configuration* file.
- **Fully-distributed approach:** In this approach, the *configuration* file can be stored in every peer of the peer-to-peer overlay network. Each peer periodically download the last version of the *configuration* file from its neighboring peers and then update its *configuration* file considering the downloaded *configuration* file.

3.2 SAN Layer

In the *SAN Layer*, the network status sensors and modifiable elements are designated based on local configurations of the peers in the peer-to-peer overlay network. In other words, each cognitive peer uses its network status sensors for gathering local information about its corresponding peer in the overlay network to observe its environment. Each cognitive peer also acts on its modifiable elements to change the local configuration of its corresponding peer in the overlay network. During design of the *SAN* layer, a list of required functions for the network status sensors and modifiable elements must be prepared. In the rest of this subsection, we describe about the required functions.

- The required functions for the network status sensors are defined based on goals of the cognitive network. For example, if delays among peers, capacity of the peers, and lifetime of the peers of the network are used in the goals of the cognitive network, the network status sensors must be able to execute appropriate functions to find local and global information about the delays among peers, capacity of the peers, and lifetime of the peers.
- The required functions for the modifiable elements are defined based on allowable actions of the management algorithms which affect on achieving the goals of the cognitive network. For example, if the management algorithm is in charge of tuning a parameter called neighborhood radius q, we must have functions for increasing (or decreasing) the value of parameter q.

3.3 Cognitive Process Layer

The *Cognitive Process Layer* is implemented using a set of cognitive engines resided in the cognitive peers. The cognitive engine consists of two layers (Fig. 7): *Configuration Manager Layer* and *Cognitive Process Manager Layer*. The descriptions of the layers of the *Cognitive Process Layer* are given bellows.

- ***Cognitive Process Manager Layer:*** This layer manages the operations of the cognitive peers using a network of cognitive processes (Fig. 8). In this layer, a network of cognitive processes observes the status of the peer-to-peer overlay network using network status sensors of the cognitive peers. The network of cognitive processes of this layer also cooperatively changes the configuration of the cognitive peers using the modifiable elements of the cognitive peers considering the goals of the cognitive network. It should be note that, the *Cognitive Process Manager Layer* is aware of the goals of the cognitive network and therefore the network of cognitive processes of the cognitive peer-to-peer network update the configuration of the cognitive peers according to the goals of the cognitive network.

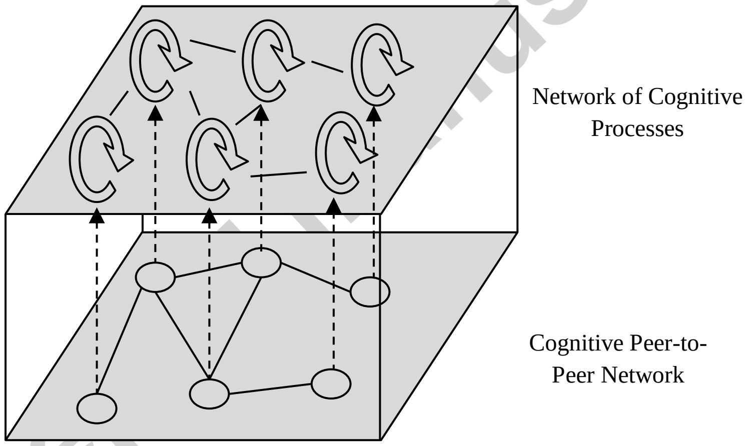
Network of Cognitive Processes
Cognitive Peer-to-Peer Network

Fig. 8. Network of Cognitive Processes

- ***Configuration Manager Layer:*** In this layer, each cognitive peer manages the configuration of its corresponding peer of the peer-to-peer overlay network. This layer has two major responsibilities described as follow. The first responsibility is to provide required information for the network of cognitive processes of the *Cognitive Process Manager Layer*. The second responsibility is to activate the network of cognitive processes and execute appropriate management algorithm determined by the cognitive processes of the *Cognitive Process Manager Layer* using SAN layer. This layer also executes other required management algorithms of the peer-to-peer overlay network which do not use the *Cognitive Process Manager Layer*.

During implementation of the *Cognitive Process Layer*, several decisions should be made which some of them are explained as below.

- Decision about selecting the goal of the cognitive network that determines which objective function should be optimized.
- Decision about selecting network status sensors considering the goal of the cognitive network.
- Decision about selecting modifiable elements which can be used in the network of cognitive processes in order to improve the performance of the peer-to-peer overlay network.
- Decision about selecting appropriate learning mechanism considering the status of the peer-to-peer network. Because of distributed and dynamic nature of peer-to-peer networks, an artificial intelligence technique which can better handles volatile, incomplete, and distributed information about nodes of the network is a good candidate for designing learning mechanism of the cognitive engines in peer-to-peer networks[1] [21].

In the next section, a model of cellular learning automata is used to implement the network of cognitive processes. Cellular learning automata are distributed self-organizing model with online learning capabilities. Note that, we can use every learning mechanism considering the conditions of the peer-to-peer networks in the proposed framework.

4 An Approach based on *CLA* for Designing Cognitive Engines and Its Application to Solve Topology Mismatch Problem

In this section, we first suggest a new model of *CLA*, an approach based on this model of *CLA* for designing cognitive engines for cognitive peer-to-peer networks, then propose a cognitive engine for solving topology mismatch problem in unstructured peer-to-peer networks, and finally present an example about the proposed cognitive engine.

4.1 Asynchronous Dynamic Cellular Learning Automata

In Asynchronous Dynamic Cellular Learning Automaton (*ADCLA*), connections between cells may be changed during the course of evolution of *CLA* to modify the cellular structure. The proposed model is described as bellow.

Definition 1. An Asynchronous Dynamic Cellular Learning Automaton (*ADCLA*) is an 9-tuple *ADCLA*= (G, A, N, Φ, Ψ, F_1,F_2, F_3, F_4), where:

- $G = (V, E)$ is an undirected graph which determines the structure of *ADCLA* where $V = \{cell_1, cell_2, \dots, cell_n\}$ is the set of vertices and *E* is the set of edges.

- $A = \{LA_1, LA_2, \dots, LA_n\}$ is a set of *LAs* each of which is assigned to one cell of *ADCLA*. The set of actions of automaton for a cell is the same as the set of states for that cell.
- $N = \{N_1, N_2, \dots, N_n\}$ 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 $cell_i$.

- $\Phi = \{\Phi_1, \Phi_2, \dots, \Phi_n\}$ where $\Phi_\mathrm{i} = \{(j, \alpha_l) | cell_j \in N_i \text{ and action } \alpha_l$ has been chosen by $LA_j\}$ denotes the state of *cell_i*. Φ_i^1 determines the state of *cell_i* when $\theta_\mathrm{i} = 1$.
- $\Psi = \{\Psi_1, \Psi_2, \dots, \Psi_n\}$, $\Psi_\mathrm{i} = \{(j, X_\mathrm{j}) \mid cell_j \in N_i\}$ denotes the attribute of $\mathrm{cell_j}$ where $X_\mathrm{j} \subseteq \{x_1, x_2, \dots, x_s\}$. $\{x_1, x_2, \dots, x_s\}$ is the set of allowable attributes. Ψ_i^1 determines the attribute of $\mathrm{cell_i}$ when $\theta_\mathrm{i} = 1$.
- $F_1: (\underline{\Psi}) \rightarrow (\underline{\zeta})$ 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 $< \Psi_i >$ and then returns a value from the closed interval [0,1] for ζ_i^1 where ζ_i^1 determines the restructuring signal of *cell_i*. Note that, $\zeta_i = \{(j, \zeta_j^1) | cell_j \in N_i\}$ is the set of restructuring signals of neighbors of *cell_i.*
- $F_2: (\underline{N}, \underline{\Psi}, \underline{\zeta}) \rightarrow (\underline{N}^1)$ is the structure updating rule. In each cell, the structure updating rule finds the immediate neighbors 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. For example, in *cell_i*, structure updating rule takes $< N_i, \Psi_i, \zeta_i >$ and returns N_i^1.
- $F_3: (\underline{\zeta}) \rightarrow (\underline{\nu})$ is the automaton trigger function. Upon activation of a cell, automaton trigger function is called to determine whether the learning automata residing in that cell to be triggered or not. If the automaton trigger function returns true then the learning automata of the cell will be triggered. The automaton trigger function in *cell_i* takes $< \zeta_i >$ and returns a value from {true, false} for ν_i where ν_i is called automaton trigger signal.
- $F_4: (\underline{\Phi}, \underline{\Psi}) \rightarrow (\underline{\beta})$ is the local rule of *ADCLA-ML*. In each cell, the local rule computes the reinforcement signal for the learning automata of that cell based on the states and attributes of that cell and its neighboring cells. For example, in *cell_i*, local rule takes $< \Phi_i, \Psi_i >$ and then computes the reinforcement signal $< \beta_i >$ for the learning automata of *cell_i*.

In each cell of *ADCLA*, the reinforcement signal is used (as the response of the environment) to change the action probability of the corresponding learning automaton and the restructuring signal is used by the structure updating rule. The application determines which cell must be activated. Upon an activation of a cell, the cell computes its restructuring signal using restructuring function (F_1) and then asks its neighboring cells to compute their restructuring signals. Then the cell calls the automaton trigger function (F_3) to find out whether its learning automaton must be triggered or not. If the learning automaton is triggered then the learning automaton selects one of its actions. The selected action of the learning automaton determines a new state for its corresponding cell. Then, the cell use the local rule (F_4) to compute the reinforcement signal to update the action probability vector of its learning automaton. If the learning automaton is not triggered, the state of corresponding cell remains unchanged during the activation of the cell. Finally, the set of neighbors of the cell is reconstructed (reconfigured and updated) using the structure updating rule (F_2). Note that, when a cell of the *ADCLA* is activated, the iteration number of the *ADCLA* increases one unit. The cells of the *ADCLA* try to changes their neighbors using the structure updating rule in order to increase the values of their restructuring signals and the learning automata of the cells try to change their actions in order to increase the values of their reinforcement signals.

Fig. 9 shows the internal structure of $cell_i$. The cell environment manager unit is in charge of executing the structure updating rule. The cell environment manager unit monitors information about local environment of $cell_i$, and the environment consisting of local environments of the cells which are in the neighborhood of $cell_i$. Note that the environment of the cell may change over time because the structure updating rule may modify the cellular structure. Another task of cell environment manager unit is to provide information about the cell and its neighbors for local rule, restructuring function, and automaton trigger function units. Note that, the proposed model of cellular learning automata can be called Closed Asynchronous Dynamic Cellular Learning Automata because it doesn't use external environment for cells. As it was previously mentioned in section 2, open cellular learning automata use external environment for cells.

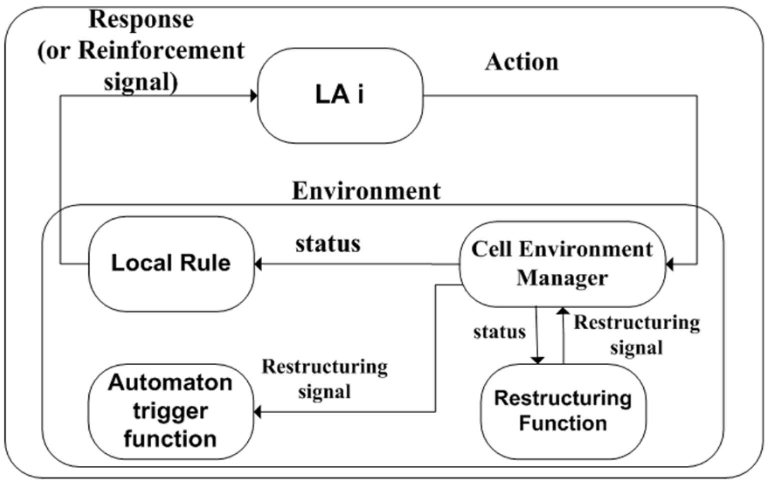

Fig. 9. Internal structure of cell$_i$

4.2 An Approach based on ADCLA for Designing Cognitive Engines

In this section, we propose an approach based on *ADCLA* for designing cognitive engines for solving network management problems in peer-to-peer overlay networks. In this approach, the *Cognitive Process Manager Layer* of the cognitive engine of the presented framework uses an *ADCLA* whose structure is isomorphic to the cognitive peer-to-peer network (Fig. 10). Each cognitive peer of the cognitive peer-to-peer network corresponds to a cell of *ADCLA*. In the *Cognitive Process Manager Layer*, executing the activation function of the *ADCLA* which results in updating the structure and the states of the cells of the *ADCLA* leads to changing the structure and the parameters of the peers of the peer-to-peer overlay network in order to solve a network management problem of the peer-to-peer overlay network. In each cell, the activation function is in charge of implementing the cognitive process of its corresponding cognitive peer. In other word, a network of cognitive processes based on *ADCLA* is conducted to solve the network management problem. Note that, in each cognitive peer, several modifiable elements are defined for changing the configurations of the corresponding peer of that cognitive peer in the peer-to-peer overlay network. In the next section, a cognitive engine based on *ADCLA* for solving the topology mismatch problem is designated.

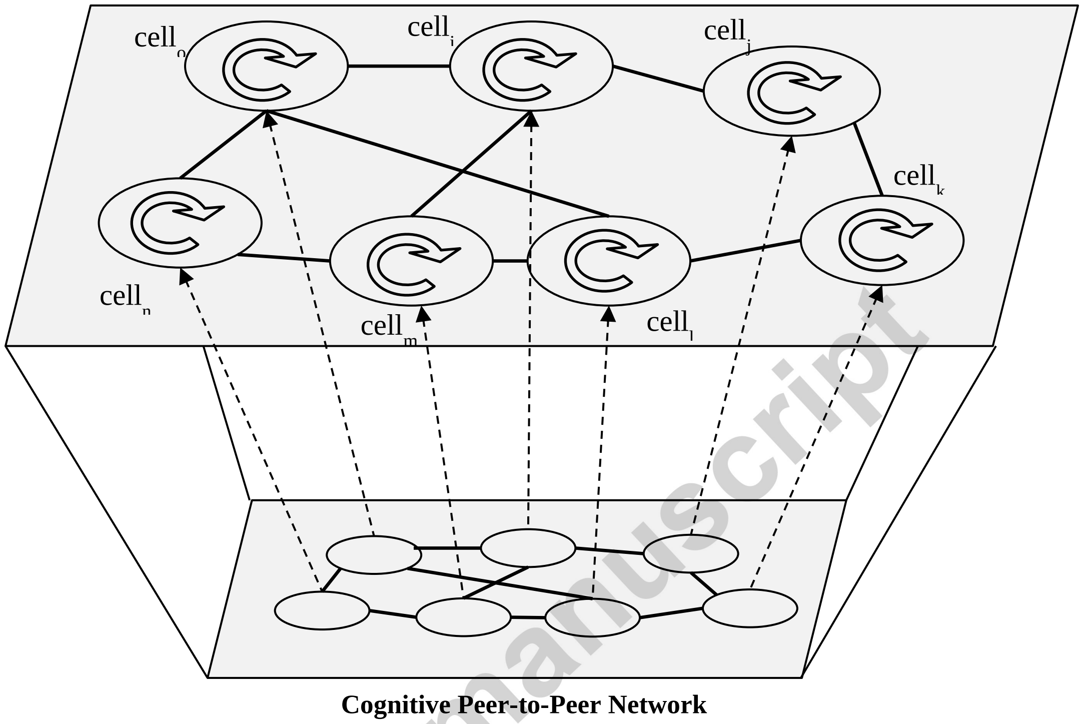

Fig. 10. A cognitive peer-to-peer network and its corresponding network of cognitive processes based on ADCLA

4.3 A Cognitive Engines for Solving Topology Mismatch Problem

In this section, a cognitive engine for solving the topology mismatch problem is designated. In the *ADCLA* of the designated cognitive engine, the rules are borrowed from *PROP-O* algorithm and *Schelling segregation* model. In the cognitive processes conducted by this cognitive engine, executing the rules of the *ADCLA* which results in updating the structure of the *ADCLA* leads to changing the structure of the peer-to-peer overlay network in order to solve the topology mismatch problem. In this cognitive engine, each cognitive peer also uses its corresponding cell to tune up the value of its parameter neighborhood radius. Parameter neighborhood radius is used in the cognitive engine. Each cell of the *ADCLA* is equipped with a learning automaton. This learning automaton which has two actions: "Increase parameter" and "Decrease parameter", is responsible for adaptively tuning up the parameter neighborhood radius.

Remark 2: The modifiable elements, network status sensors, and network of cognitive processes of the proposed cognitive engine are explained as below.

- Modifiable elements are able to change two things: the value of parameter neighborhood radius and the list of neighbors.
- Network status sensors are able to gather information about delays among peers. Note that, the goal of the cognitive network is to minimize (3) subject to constraints (4) and (5).
- The network of cognitive processes is executed by the *ADCLA* in order to solve topology mismatch problem. The rules borrowed from *PROP-O* algorithm enables the *ADCLA* to solve the topology mismatch problem. As it was previously mentioned, *PROP-O* algorithm suffer from two problems; neither the neighborhood radius nor the exchange operator can adapt themselves to the dynamicity of the network. For solving these problems, the *ADCLA* of the uses learning automata to adapt neighborhood radiuses of peers and rules borrowed from *Schelling segregation* model to adaptively manage the execution of the exchange operator. In each cognitive peer, several modifiable elements are defined for changing the neighborhood radius, and changing the neighbors of that cognitive peers. Note that, changing the neighbors of each cognitive peers leads to changing the neighbors of the corresponding peer of that cognitive peer in the peer-to-peer overlay network.

Now we present the detailed descriptions of *Cognitive Process Manager Layer* and *Configuration Manager Layer* of the proposed cognitive engine. The descriptions of these layers are given in the next two subsections.

4.3.1 Cognitive Process Manager Layer:

This layer manages a network of cognitive processes based on *ADCLA*. In each cognitive peer, a cognitive process is executed by the activation function of its corresponding cell. In the rest of this section, at first, the required variables are defined, then the design of components of the *ADCLA* are given, and finally the activation function of the *ADCLA* is described.

Required variable of the *ADCLA*: In the proposed cognitive engine, each cognitive peer contains the data structure and algorithms of a cell of *ADCLA*. For example, $peer_i$ contains the data structure and algorithms of $cell_i$. Therefore each cognitive peer has variables for saving the attribute, state, neighbors, and the probability vector of learning automaton of its corresponding cell. In addition. The following items are defined for designing the components of the *ADCLA*.

- Neighborhood radius of $peer_i$ is denoted by r_i.
- Neighborhood set of $peer_i$ denoted by ${NP_i}^r$ contains all peers residing in the neighborhood radius of $peer_{i,}$ that is ${NP_i}^r = \{\, peer_j \in V \mid dist(peer_i, peer_j) \leq r_i\}$ where $dist(peer_i, peer_j)$ is the length of the shortest path (with minimum number of hops) between $peer_i$ and $peer_j$ in the overlay network. The immediate neighbors of $peer_i$ is denoted by ${NP_i}^1$.
- Candidate peer set of $peer_i$ for $peer_j$ denoted by C_{ij}. This set contains some of the neighbors of $peer_i$ which change their connections from $peer_i$ to $peer_j$ during the exchange operation.

- Corresponding peer set of $peer_i$ denoted by MP_i. This set contains some of neighbors of $peer_i$ such as $peer_j \in NP_i{}^r$ which $peer_j$ has a candidate peer set $C_{ji} \subseteq NP_j{}^1$ and $peer_i$ has a candidate peer set $C_{ij} \subseteq NP_i{}^1$ such that $|C_{ji}|=|C_{ij}|$ and

$$\left[\sum_{peer_k \in C_{ij}} d_{ki} + \sum_{peer_l \in C_{ji}} d_{lj}\right] > \left[\sum_{peer_l \in C_{ji}} d_{li} + \sum_{peer_k \in C_{ji}} d_{kj}\right].$$

  In other words, MP_i contains some of the neighbors of $peer_i$ which can participate in the exchange operation to decrease the delays of paths and also mismatched paths of the overlay network.

- $\lambda_i = \frac{|NP_i{}^1| - |MP_i|}{|NP_i{}^1|}$ is the portion of neighboring peers of $peer_i$ which do not have mismatched path with $peer_i$ (the portion of the neighbors of $peer_i$ which cannot participate in the exchange operation with $peer_i$). λ_i is called similarity function. The value of λ_i increases during the exchange operation and reaches 1 when $peer_i$ has no mismatch path ($|MP_i| = 0$). Function λ_i gives valuable information about the neighbors of $peer_i$ and it is used to tune some components of *ADCLA* which implement the process of changing neighbors of cells.

In the *ADCLA*, each cell has two states: "Increase parameter"and "Decrease parameter". The initial state of all cells is set to "Increase parameter". Each cell is equipped with a learning automaton which has two actions: "Increase parameter" and "Decrease parameter" to determines the state of that cell. In $peer_i$, the state of $cell_i$ will be used to increase (or decrease) the value of variable r_i. According to the definition of *ADCLA*, we need to determine attributes of cells to use them in restructuring function, structure updating rule, and local rule. But here, attributes cannot be computed due to the fact that peers are not aware of their underlay positions. This does not cause any problems due to the fact that the proposed model of *ADCLA* uses the similarity function (λ_i) which does not need to use attributes of cells. The similarity function gives required information about the neighbors of the peer.

Design of components of the *ADCLA*: The components of the *ADCLA* are described as below.

- **Restructuring function** takes information about neighbors of a cell $cell_i$ as input and then returns a restructuring signal. In $cell_i$, the restructuring signal $\zeta_i^1 = \lambda_i - \frac{1}{|NP_i{}^1|}$.

- **Structure updating rule** is implemented using an operator called swap operator. Fig. 11 shows an example of usage of swap operator. In this figure, if $\zeta_i^1 < z$ (parameter *z* is initially set to a given value) then the structure updating rule selects $cell_j$ using a function called *prop-selector ()* and then exchanges equal number (determined by parameter m which is initially set to a given value) of neighbors between $cell_i$ and $cell_j$ in order to increase the value of ζ_i^1. In $cell_i$, function *prop-selector()* takes information about $cell_i$, information about a neighboring cell called *s-cell*, and the neighbors of $cell_i$ determined by the neighborhood radius r_i and then returns one cell as output. The detailed descriptions about the structure updating rule and also function *prop-selector()* are given later.

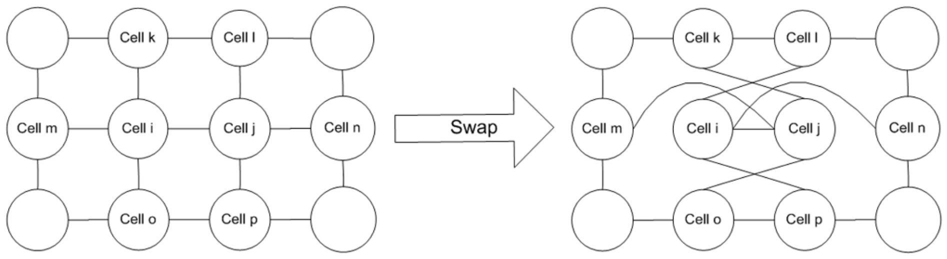

Fig. 11. an example of Swap operator of the ADCLA

- **Automaton trigger function** takes the restructuring signal of a cell *cell*$_i$ as input and then returns true if the value of the restructuring signal of *cell*$_i$ or one of neighbors of *cell*$_i$ is lower than z and false otherwise.

- **Local rule** takes information of immediate neighbors of a cell *cell*$_i$ as input and then returns a reinforcement signal (β_i) for the learning automaton of *cell*$_i$. The output of the local rule of *cell*$_i$ is equal to 1 in only two cases described as follows. In the first case, the value of λ_i is higher than a parameter t (parameter t is initially set to a given value) and the action selected by the learning automaton of *cell*$_i$ is equal to ”Decrease parameter”. In the second case, the value of λ_i is lower than or equal the parameter t and the action selected by the learning automaton of *cell*$_i$ and the majority of states of immediate neighboring cells of *cell*$_i$ are equal to ”Increase parameter”. In other cases, the local rule returns 0.

Remark 3: The ideas according to which the structure updating rule is obtained are borrowed from *PROP-O* algorithm and *Schelling segregation* model which are described as bellows.

- The swap operator of *ADCLA* implements the exchange operator of *PROP-O* algorithm.
- The definition of restructuring function is borrowed from the similarity function of *Schelling segregation* model. The goal of increasing the value of similarity function in the *Schelling segregation* model is the same as the goal of increasing the value of restructuring signal in the *ADCLA*.
- In the *ADCLA*, similar to the *Schelling segregation* model, *cell*$_i$ changes its neighbors using the structure updating rule in order to increase the portion of neighboring cells that they have similar attribute with *cell*$_i$ which leads to increasing the value of restructuring signal ζ_i^1.
- Function *prop-selector()* is designated based on the local search of *PROP-O* algorithm. The process of finding appropriate cell for executing swap operation in the *ADCLA* plays the same role as the process of finding a *corresponding* peer for executing exchange operation in the *PROP-O* algorithm. For example, in *cell*$_i$, *prop-selector()* returns *cell*$_j$ which *peer*$_j$ considering the local search of *PROP-O* algorithm is an appropriate *corresponding* peer for executing exchange operation with *peer*$_i$. Cell *s-cell* which is used as input in *prop-selector()* denotes the cell of the peer which is used to start the *local search* of *PROP-O* algorithm.

Design of Activation function of *ADCLA*: Fig. 12 shows the pseudo code of the function which each cell of *CLA* executes after activation. After activating a *cell*$_i$, the *cell*$_i$ gathers required information about neighboring cells, uses restructuring function to compute the restructuring signal ζ_i^1, and then uses automaton trigger function to compute the automaton trigger signal v_i. If the automaton of *cell*$_i$ is triggered then *peer*$_i$ asks learning automaton of *cell*$_i$ to choose an action, uses the selected action to set the value of Φ_i^1 (which determines the state of *cell*$_i$), computes the reinforcement signal using local rule, and then updates the learning automaton of *cell*$_i$. If the learning automaton of *cell*$_i$ isn't triggered then the value of Φ_i^1 remains unchanged. Then, *cell*$_i$ gathers information about neighboring cells and tries to execute structure updating rule of *cell*$_i$. After execution of structure updating rule the list of neighbors of *cell*$_i$ will be modified. The neighbors of *cell*$_i$ cooperate with *cell*$_i$ for exchanging information and executing swap operation. It should be noted that, the *Configuration Manager Layer* which will be described in the next part uses function *activate_cell()* of the *Cognitive Process Manager Layer* to activate its corresponding cell.

```
Algorithm activate_cell()
        Inputs:
            z// the parameter for the portion of the neighbors of
                the cell that have similar attribute with the cell.
                parameter z is used by the automaton trigger
                function and the structure updating rule of the
                cell.
            t// the parameter for the portion of the neighbors of
                the cell that have similar attribute with the cell.
                parameter t is used by the local rule of the cell.
            r// the parameter neighborhood radius of the cell.
            m// the number of cells which can be exchanged
                among cells.
            s-cell //the cell which is used to find other
                    neighboring cells.
01      Begin
02       - gather information of the neighboring cells;
03       - compute the restructuring signal of the cell using
           the restructuring function ($F_1$);
04       - call the automaton trigger function($F_3$) to
           determine the value of automaton trigger signal;
           //the automaton trigger function uses parameter z
05       If (the value of automaton trigger signal is equal to
              true)Then
06            - ask from learning automaton of the cell to
              chose an action;
07            - set the state of the cell to the action chosen
              by the learning automaton;
08            - compute the reinforcement signal using the
              local rule($F_4$);//the local rule uses parameter t
09            - update the learning automaton of the cell
               using computed reinforcement signal;
10       EndIf
11       - find the immediate neighbors of the cell using
```

```
            the structure updating rule ( $F_2$ );
            //the structure updating rule uses
              two parameters z and m,
              parameter neighborhood radius r
              , and cell s-cell
12      - determine the neighbors of the cell using the
          output of the structure updating rule;
13      End
```

Fig. 12. the pseudo code of the procedure which each cell executes after activation

4.3.2 Configuration Manager Layer

This layer utilizes the *Cognitive Process Manager Layer* to change the connections of the peers of the peer-to-peer overlay network in order to solve the topology mismatch problem. In each peer, the algorithm which is used in this layer starts with *extended-warm-up* phase and ends with *extended-maintenance* phase. In the rest of this part, the phases of this algorithm are described.

In the *extended-warp-up* phase, $peer_i$ gathers information about its neighboring peers, selects one of its neighbors called $peer_s$, and calls *activate_cell()* function from *Cognitive Process Manager Layer*. Then $peer_i$ changes the value of its neighborhood radius r_i according to the state of $cell_i$ (the cell of $peer_i$). Note that when the state of $cell_i$ is equal to "Increase parameter" ("Decrease parameter") we increase (decrease) the value of variable r_i one unit. If the value of neighborhood radius is lower than 1, $peer_i$ set the value of neighborhood radius to 1. Finally, $peer_i$ refines the list of its neighbors considering the management operations executed by the *activate_cell()* function of the *Cognitive Process Manager Layer*, and then decides to repeat *extended-warm-up* phase or not using a counter called *ntrial*. Counter *ntrial* is use to control the iterations of the extended-warm-up phase. The pseudo code of the *extended-warp-up* phase is given in Fig. 13.

In the *extended-maintenance* phase, $peer_i$ gathers information about its neighboring peers, selects one of its neighbors called $peer_s$, and calls *activate_cell()* function from *Cognitive Process Manager Layer*. Then $peer_i$ changes the value of its neighborhood radius r_i according to the state of $cell_i$ (the cell of $peer_i$). Note that when the state of $cell_i$ is equal to "Increase parameter" ("Decrease parameter") we increase (decrease) the value of variable r_i one unit. If the value of neighborhood radius is lower than 1, $peer_i$ set the value of neighborhood radius to 1. If the neighbors of the $peer_i$ has been changed, $peer_i$ refines the list of its neighbors, decreases the priority of $peer_s$. Otherwise, the priority of the selected neighbor $peer_s$ will be decreased and the timer of the $peer_i$ will be changed. After updating the *neighborQ*, $peer_i$ decides to repeat the *extended-maintenance* phase or not using a Timer. In the *extended-maintenance* phase, the matching process will be repeated till the end of a duration computed based on two parameters *MAX_TIMER* and *INIT_TIMER*. In the *extended-maintenance* phase, each peer resets its neighborhood radius and the variables of its corresponding cell if it receives join or leave messages or when its connections changes. The pseudo code of the *extended-maintenance* phase is given in Fig. 14.

Remark 4: the algorithm used in the *Configuration Manager Layer* is a modified version of *PROP-O* algorithm. The phases of this algorithm are modified version of *warm-up* phase and *maintenance* phase of *PROP-O* algorithm respectively. Some parts of this algorithm such as the algorithms of selecting neighboring peers, selecting a peer for starting the matching procedure (called *peer*$_s$), setting the timers and changing the priority of peers in *neighborQ* are similar to *PROP-O* algorithm.

```
Algorithm extended-warm-up()
        Input:
            r // the initial value of parameter neighborhood radius
            INIT_TIMER// the initial value for variable timer in the algorithm
            MAX_INIT_TRIAL // the maximum number of iteration in the
                                    algorithm
            z,t, and m // the parameters used by the cell of the peer

        Notations:
            Let timer denotes a waiting time which is used after each exchange
            operation.
            Let neighborQ denotes a priority queue which saves the information
            about neighbors of the peer.
            Let ntrial denotes the iteration number at each iteration.
01      Begin
02      - timer ← INIT_TIMER;
03      - add all neighbors into neghborQ;
04      While ntrial < $MAX\_INIT\_TRIAL$ do
05          - s ← neighborQ.pop;
06          - neighborQ.addTail(s);
07          - make peer_s as destination of the first hop;
              // s-cell denotes the cell of peer_s
08          - gather required information for the cell of the peer;// prepare the cell
        for
                                                executing function
                                                activate_cell
09          - call activate_cell(z, t, r, m, s-cell);
              //the pseudo code of this function is given in Fig.  12
10          If (the state of the cell of the peer is equal to "Increase
                parameter") Then
11              - $r \leftarrow r + 1$;
12          Else
13              - $r \leftarrow r - 1$;
14          EndIf
15          If (r<1) Then
16              - $r \leftarrow 1$;
17          EndIf
18          - refine the list of neighbors of the peer;
            // considering the operation executed by the cell of the peer
19          - ntrial ← ntrial + 1
20          - wait timer before next ntrial;
21      EndWhile
22      End
```

Fig. 13. Pseudo code of extended-warm-up phase of *PROP-OL* algorithm

```
    Algorithm extended-maintain()
        Input:
            r// the initial value of parameter neighborhood radius
            INIT_TIMER// the initial value for variable timer in the algorithm
            MAX_TIMER // the maximum value for waiting time
            t,z, and m // the parameters used by the cell of the peer
```

Notations:

Let **timer** denotes a waiting time which is used after each exchange

Let **neighborQ** denotes a priority queue which saves the information about neighbors of the *peer*.

```
01 Begin
02 While time expires do
03     - $s \leftarrow \text{neighbor.pop}$;
04     - make $peer_s$ as destination of the first hop;
05     - gather required information for the cell of the peer;// prepare the cell for
                                                     executing
   function
                                                     activate_cell
06     - call activate_cell(z, t, r, m, s-cell);
       //the pseudo code of this function is given in Fig. 12.
07     If (the state of the cell of the peer is equal to "Increase
           parameter") Then
08         - $r \leftarrow r + 1$;
09     Else
10         - $r \leftarrow r - 1$;
11     EndIf
12     If (r<1) Then
13         - $r \leftarrow 1$;
14     EndIf
15     If (the neighbors of the peer has been changed) Then
16             - refine the list of neighbors of the peer;
               // according to the changes made by the cell of the peer
17             - $\text{s.priority} \leftarrow \text{s.priority} - 1$;
18     Else
19       -$\text{s.priority} \leftarrow \text{neighborQ.minPriority} - 1$;
20        - $\text{timer} \leftarrow \min(\text{timer} * 2, \text{MAX\_TIMER})$;
21        EndIf
22        - refresh neighborQ;
23 EndWhile
24 If (receive join/leave message or detect failure entries) Then
25        - $\text{timer} \leftarrow \text{INIT\_TIMER}$;
26        - add new entries to the front of neighborQ;
27        - reset variables of the cell of the peer;
28 EndIF
29  End
```

Fig. 14. Pseudo code of extended-maintenance phase of *PROP-OL* algorithm

Remark 5: The rationale behind of designing the network of cognitive processes is described as follows. The process of changing neighbors in the network of cognitive processes which is based on *PROP-O* algorithm plays the same role as the process of changing neighbors of agents in the *Schelling segregation* model. Fig. 15 shows the process of changing neighbors in both *Schelling segregation* model and *ADCLA*. Fig. 16 shows the network of cognitive processes and its corresponding Schelling segregation model and also illustrates the neighbors of $cell_i$ when the neighborhood radius is equal to 1.

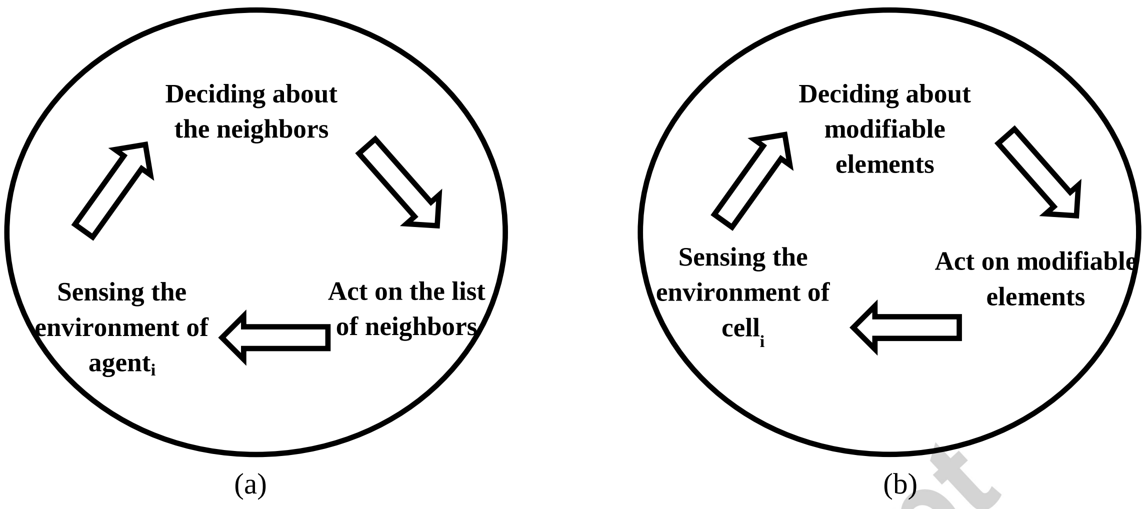
Deciding about the neighbors
Act on the list of neighbors
Sensing the environment of agent$_i$
(a)
Deciding about modifiable elements
Act on modifiable elements
Sensing the environment of cell$_i$
(b)

Fig. 15. (a) the process of changing neighbor in agent$_i$ of Schelling segregation model (b) the process of changing neighbor in cell$_i$ of ADCLA

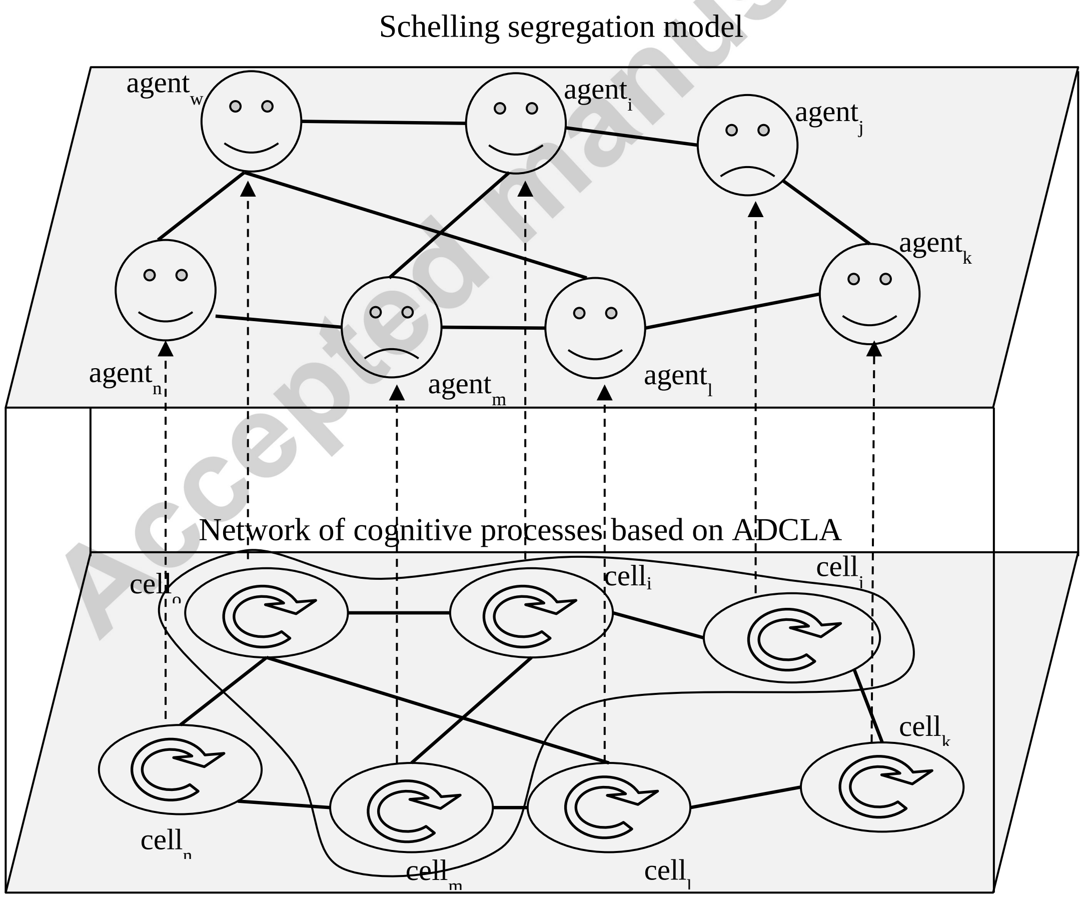

Fig. 16. Schelling segregation model and its corresponding network of cognitive processes

As it was previously mentioned, *PROP-O* algorithm suffer from two problems; neither the neighborhood radius nor the exchange operator can adapt themselves to the dynamicity of the network. In the network of cognitive processes of the proposed cognitive engine, these problems were solved with the following solutions.

- The similarity function of *Schelling segregation* model was used to manage the execution of exchange operation. In the network of cognitive processes, similar to the *Schelling segregation* model, $peer_i$ changes its neighbors in order to increase the portion of neighboring peers (λ_i) which do not have mismatched path with $peer_i$. Since the value of similarity function gives valuable information about the position of $peer_i$, it is used in the stopping condition of the structure updating rule of the *ADCLA* which manages the execution of exchange operation. In other word, the cognitive processes conducted by the *ADCLA* observes the information about the positions of the peers in order to adaptively manage the exchange operator considering the current status of the network.

- The learning automata of the *ADCLA* are used to manage the neighborhood radiuses of the peers. In the network of cognitive processes, learning automata are used to find appropriate values for the neighborhood radiuses of the peers. The current status of the network determines appropriate values for the neighborhood radiuses. Therefore, in each peer, the local rule of the ADCLA which is in charge of generating reinforcement signal of the learning automaton of that peer utilizes the value of similarity function and also information about the neighboring peers of that peer.

4.4 An example

In this section, we illustrate the proposed algorithm step by step for a sample peer-to-peer network with 4 peers (Fig. 17). This peer-to-peer network is composed of 4 peers ($peer_i$, $peer_j$, $peer_k$, $peer_l$) and 3 links (($peer_i$, $peer_l$), ($peer_j$, $peer_k$), ($peer_i$, $peer_j$)). This network is constructed over an underlay network that comprises 4 positions (pos_i, pos_j, pos_k, pos_l) and 3 links ((pos_i, pos_l), (pos_j, pos_l),(pos_k, pos_l)). In this example, to transfer a message from $peer_i$ to $peer_k$ in the peer-to-peer network path ($peer_i \rightarrow peer_j \rightarrow peer_k$) is used, but in the underlay network, we may have an inefficient path such as ($pos_i \rightarrow pos_l \rightarrow pos_j \rightarrow pos_l \rightarrow pos_k$). This means that a message may meet an underlay position several times that causes the communication delay and traffic of flooding techniques to increase. In this example, path ($peer_i \rightarrow peer_k$) is called a mismatched path. Therefore this peer-to-peer network suffers from topology mismatch problem. It should be noted that, the positions of the peers are fixed, and in order to solve the mismatch problem, we can change the links of the peer-to-peer networks.

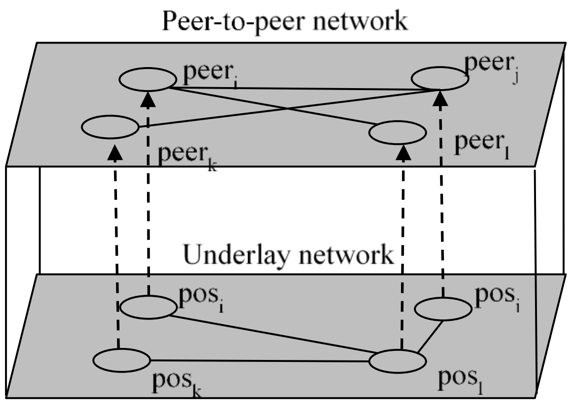

Fig. 17. A sample peer-to-peer network

The proposed algorithm, creates a cognitive peer-to-peer network that is composed of 4 cognitive peers (*cpeer*$_i$, *cpeer*$_j$, *cpeer*$_k$, *cpeer*$_l$) and 3 links ((*cpeer*$_i$, *cpeer*$_l$),(*cpeer*$_j$, *cpeer*$_k$),(*cpeer*$_i$, *cpeer*$_j$)). Since each cognitive peer uses a cell in its cognitive engine, a network of cells which construct the *ADCLA* is built over the cognitive peer-to-peer network. This *ADCLA* is composed of 4 cells (*cell*$_i$, *cell*$_j$, *cell*$_k$, *cell*$_l$) and 3 links ((*cell*$_i$, *cell*$_l$),(*cell*$_j$, *cell*$_k$),(*cell*$_i$, *cell*$_j$)). Fig. 18 shows the cognitive peer-to-peer network of the peer-to-peer network that is illustrated in Fig. 17.

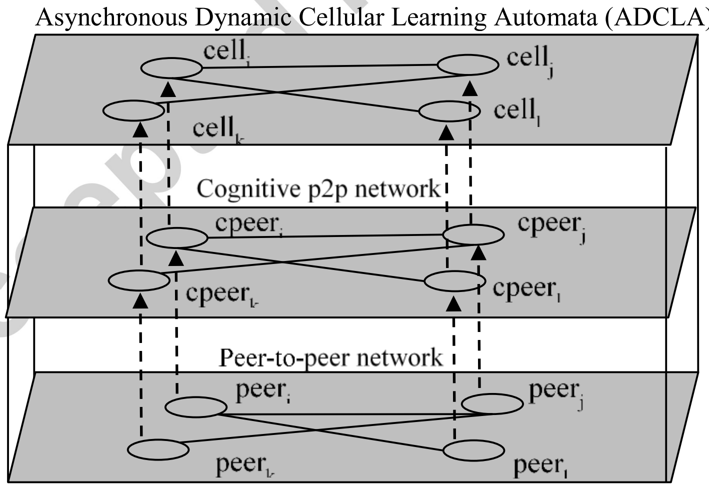

Fig. 18. The cognitive peer-to-peer network of the peer-to-peer network that is illustrated in Fig. 17

A possible execution of the proposed cognitive engine for *peer*$_i$ is shown in Fig. 19 step by step. In Fig. 19, peer$_i$ is selected and colored with black to activate the proposed cognitive engine. In this example, let the neighborhood radius of *peer*$_i$ is equal to 1, and the parameters z and t of the *ADCLA* are set to 1 and 0.6 respectively.

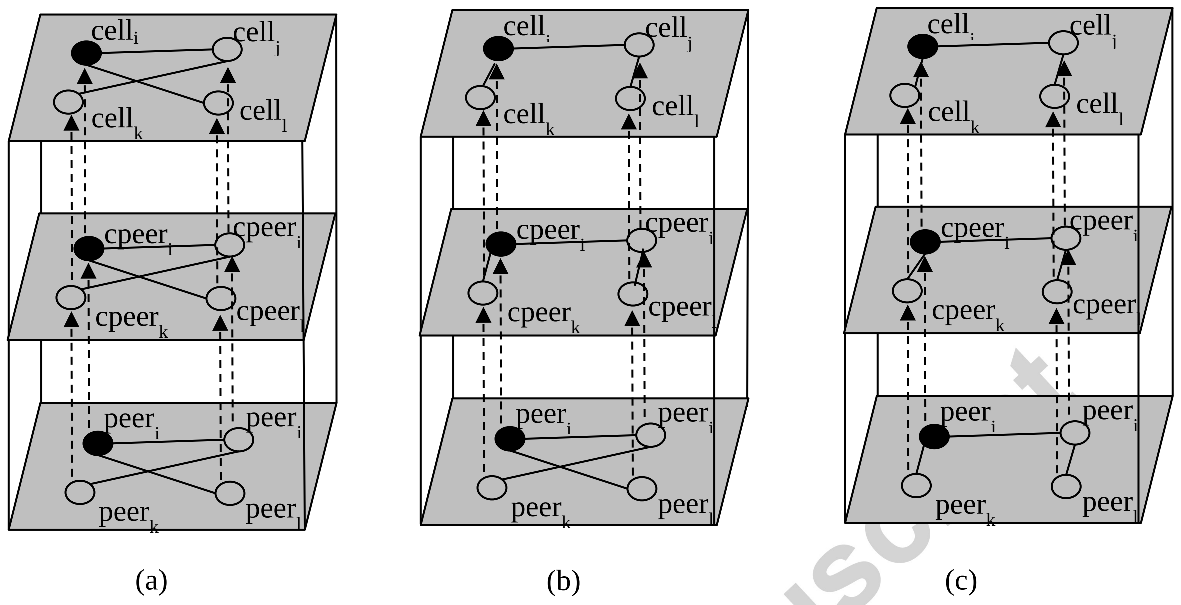

Fig. 19. The steps of a possible execution of the proposed cognitive engine for *peer*$_i$

Fig. 19 (a) shows the network before execution of the proposed cognitive engine in *peer*$_i$. The variables used in the *peer*$_i$ are shown in Table 1.

Table 1. The variables of the *peer*$_i$ before execution of the proposed cognitive engine

	Variables	Values
Variables of *cell*$_i$	θ_i	1
	N_i	{*cell*$_i$, *cell*$_j$, *cell*$_l$}
	ζ_i	$\frac{1}{3}$
	ν_i	True
	Φ_i	"Increase Radius"
Variables of *peer*$_i$	r_i	1
	NP_i	{*peer*$_i$, *peer*$_j$, *peer*$_l$}
	λ_i	$\frac{2}{3}$

As it was previously mentioned, the proposed cognitive engine activates the *ADCLA* to tune the configuration of the cognitive peer-to-peer network. Fig. 19 (b) show the network after execution of function activate() of *peer*$_i$. Since the automaton trigger signal is equal to true, the learning automaton of *cell*$_i$ chooses an action according to its probability vector. Let action "Increase Radius" was chosen by the learning automaton. The chosen action is used to set the state of *cell*$_i$. Then the *cell*$_i$ compute the restructuring signal and update its learning automaton. . Finally the structure updating rule changes the connections of *cell*$_i$. After execution of function activate(), the configuration of the *cell*$_i$ is used to change the configuration of the *peer$_i$*. Fig. 19 (c) shows the network after execution of the cognitive engine of *peer$_i$*. Note that, in Fig. 19 (c), the mismatched path of the network was removed.

5 Experimental Results

All simulations have been implemented using *OverSim* simulator [65]. This simulator supports different kinds of underlay network models: *Simple*, *SingleHost*, *INET*, and *ReaSE*. *Simple model* is the most scalable one and *ReaSE model* is able to generate different types of underlay networks. We used *ReaSE model* to generate router-level topologies [66]. Experiments reported in this section are conducted on three different underlying networks: *Topology.1* as an example of *Simple model*, *Topology.2* and *Topology.3* as examples of *ReaSE model* as given in Table 2. For overlay network *Gnutella* [67] which is an unstructured peer-to-peer network is used. We used the Gnutella dataset of [68] to generate the overlay topology. This data set contains a graph which its nodes, edges, and diameter are 10876, 39994, and 9 respectively. Random churn model [65] and Pareto churn model [65] are also used to model the process of joining and leaving peers in the network. In Table 3, the probabilities of joining and leaving the peers for one Random churn model and the lifetime mean and the dead time mean for one Pareto churn model used in this section are given.

Table 2. Underlay topologies

Underlay topology	Descriptions
Topology.1	In this underlay topology, peers are placed on a N-Dimensional Euclidean space and the Internet latencies are based on CAIDA/Skitter [69], [70] data.
Topology.2	Consists of 10 autonomous systems, and about 1087 router -level nodes. This topology contains few and populated groups
Topology.3	Consists of 50 autonomous systems, and about 217 router-level nodes. This topology contains many low populated groups in which the distance between the groups is far greater than the distance between the peers in each group.

Table 3. Churn model setting

Churn model type	Churn model name	Churn model parameter
Random churn model [65]	Random churn	joining_probability=0.8 leaving _probability=0.2
Pareto Churn model [65]	Pareto Churn	LifetimeMean: 90 sec DeadtimeMean:10 sec

To evaluate the performance of the proposed cognitive engine which we called *PROP-OL* it is compared with *PROP-O* algorithm [40] and *X-BOT* algorithm [46]. The reasons for choosing *PROP-O*, and *X-BOT* for the sake of comparisons is described as follows. The first reason is that, the proposed cognitive engine like algorithms *PROP-O* and *X-BOT* with which the proposed cognitive engine is compared uses only delays of links for solving topology matching problem. The second reason is that, both *PROP-O* and *X-BOT* algorithms like the proposed cognitive engine use a management operator (called exchange operator) to exchange connections among the peers and try to preserve the number of connections of peers during the matching process. It should be noted that because of using the exchange operator the number of connections of the peers remain unchanged.

Table 4. The parameters of *PROP-OL* and *PROP-O* algorithms

Parameters	Value
M	3
MIN_VAR	0
MAX_INIT_TRIAL	10
INIT_TIMER	10 min
MAX_TIMER	1000 min

Table 5. The parameters of *X-BOT* algorithm

Parameters	Value
K	6
PBO	10s
Π	2
M	3

Table 6. The parameters of the learning automata

Parameters	Value
reward parameter α	0.25
penalty parameter b	0.25

The parameters of *PROP-OL* and *PROP-O* algorithms are set based on Table 4. The parameters of *X-BOT* algorithm are set based on Table 5. For all experiments, each peer is equipped with a variable structure learning automaton of type L_{RP} with parameters reported in Table 6. Based on our experimental study using different parameter settings, we have chosen the best parameters values for PROP-O and X-BOT algorithms. The design of experiments is given as follow. Experiment 1 and experiment 2 are designated to find appropriate values for two parameters *t* and *z* for the proposed cognitive engine. From experiment 3 to experiment 5 the proposed cognitive engine is compared with *PROP-O* and X-BOT algorithms. Note that, some rules of *CLA* of the proposed cognitive engine are borrowed from *PROP-O* algorithm. Simulations are performed for 100 rounds. Results reported are averages over 50 different runs. The algorithms are compared with respect to five metrics: Overlay Communication Delay (OCD), Overlay Reconfiguration Overhead (ORO), Control Message Overhead (CMO), Mismatched Paths Level (MPL), and Mismatched Paths Delays (MPD). These metrics are briefly explained below:

- **Overlay Communication Delay (OCD)** is the sum of end-to-end delays of links in the overlay network (using (3)).
- **Overlay Reconfiguration Overhead (ORO)** is the number of peers which are reconfigured by matching algorithm. This metric implicitly shows the changes which were made by the local operators (such as exchange operator) of the matching algorithms.

- **Control Message Overhead (CMO)** is the number of extra control messages generated by the matching algorithm of the overlay network.
- **Mismatched Paths Level (MPL)** is the number of positions which are re-visited in mismatched paths of the overlay network. A mismatched path is a path in overlay network such that in its corresponding underlay path a particular position has been visited more than once.
- **Mismatched Paths Delays (MPD)** is the sum of delays of mismatched paths of the overlay network.

Experiment 1:

This experiment is conducted to study the effect of parameter *t* on the performance of the proposed cognitive engine when parameter *z* is set to 1 and parameter r is set to 2. For this study, the proposed cognitive engine is tested for three initial values for parameter *t* = 0.3, 0.6 and 0.9, and *Topology.1* is used as the underlay network. The results are compared with respect to five above mentioned criteria. According to the results of this experiment that are shown in Fig. 20 to Fig. 24 and Table 7 one may conclude the following:

- In terms of *OCD*, *MPL*, and *MPD*, the proposed cognitive engine with *t=0.6* and *t=0.9* performs better than the proposed cognitive engine with *t=0.3*. This is because low value for parameter *t* leads to low sensitivity of the cognitive engine of a peer to the information about other peers of the network. It should be noted that, in each peer, parameter *t* implicitly determines the information about neighboring peers can be used for calculating the reinforcement signal of the learning automaton in the cognitive engine or not. The information about the neighbors of peers gives valuable information about the state of the network and therefore low sensitivity of the cognitive engines conduct the learning automata of cognitive engine to converge to inaccurate actions which lead to low accuracy of the proposed cognitive engine (with respect to high *OCD*, *MPL*, and *MPD*).
- Increasing the value of parameter *t* leading to increasing *ORO* and *CMO* but decreasing *OCD*, *MPL* and *MPD* at early rounds of the simulation. In other word, by increasing the value of parameter *t* we can find an appropriate overlay with low *OCD*, *MPL*, and *MPD* but we will have high overhead (with respect to high *ORO* and *CMO*) at early rounds of the simulation.
- In terms of *ORO* and *CMO*, the proposed cognitive engine with *t=0.6* performs better than the proposed cognitive engine with *t=0.3* and *t=0.9* except for the early rounds of the simulation.
- The proposed cognitive engine when *t=0.9* performs well with respect to *OCD*, *MPL*, and *MPD*, but does not perform well with respect to *ORO* and *CMO*. This means that, for high value for parameter *t*, when the overlay topology is transformed to an appropriate overlay (with respect to low *OCD*, *MPL*, and *MPD*) the overhead of the proposed cognitive engine remains high (with respect to high *ORO and CMO*). High value for parameter *t* leads to high sensitivity of the learning automata of cognitive engines to information about other peers of the network. Since the information about the neighbors of some peers may not reflect the whole state of the network high sensitivity of the cognitive engine may conduct the learning automata of cognitive engines to

set the neighborhood radius of the peers to large values. It is obvious that large values for parameter neighborhood radius lead to creating many control messages and performing many unnecessary changes in the network which leads to high *ORO* and *CMO*.

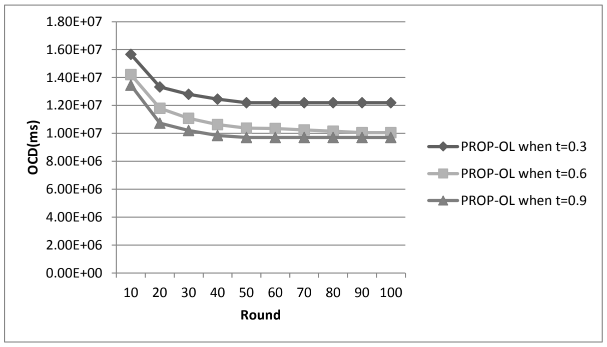

Fig. 20.The impact of parameter *t* on the performance of *PROP-OL* with respect to *OCD* when parameter *z* is1

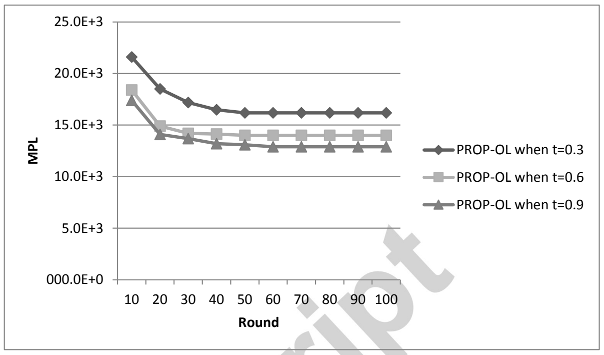

Fig. 21.The impact of parameter *t* on the performance of *PROP-OL* with respect to *MPL* when parameter *z* is1

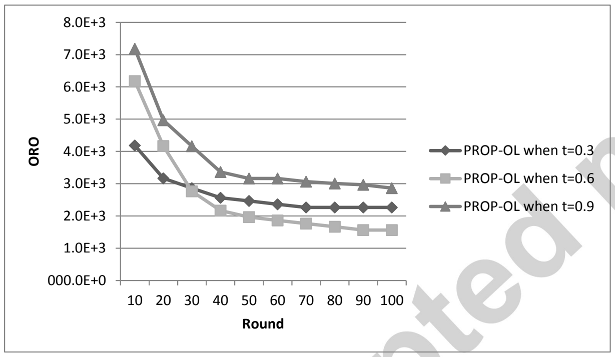

Fig. 22.The impact of parameter *t* on the performance of *PROP-OL* with respect to *ORO* when parameter *z* is1

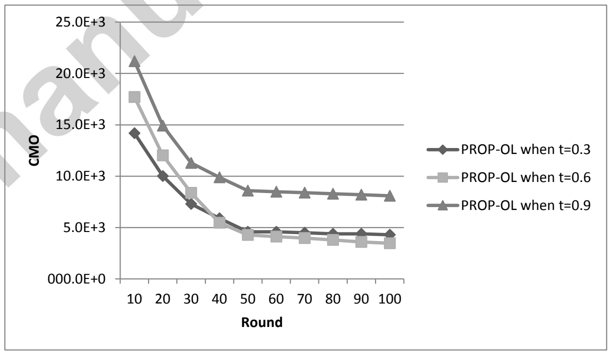

Fig. 23. The impact of parameter *t* on the performance of *PROP-OL* with respect to *CMO* when parameter *z* is1

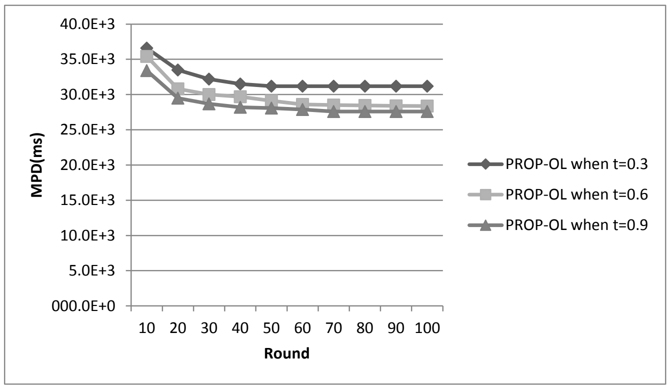

Fig. 24. The impact of parameter *t* on the performance of *PROP-OL* with respect to *MPD* when parameter *z* is1

Table 7. The required number of rounds to the lowest values of OCD for PROP-OL when z=1

	PROP-OL when t=0.3	PROP-OL when t=0.6	PROP-OL when t=0.9
The required number of rounds	34	73	42
The lowest value of OCD	1.22E+07ms	1.02E+07ms	9.70E+06ms

Experiment 2:

This experiment is conducted to show the impact of the parameter *z* on the performance of the proposed cognitive engine when the parameter *t* is set to 0.6 and parameter r is set to 2. For this purpose the proposed cognitive engine is tested for three values for parameter *z* = 0.5, 0.8 and 1, and *Topology.1* is used as underlay topology. The results are compared with respect to *OCD*, *MPL*, *CMO*, *ORO*, and *MPD*. According to the results of this experiment that are shown in Fig. 25 to Fig. 29 and Table 8 one may conclude that increasing the value of parameter *z* leading to improving the performance of the proposed cognitive engine in terms of *OCD*, *MPL*, and *MPD*. In each peer, parameter *z* implicitly activates the learning automaton of the cognitive engine. High value of parameter z leading to increasing the activity of learning automata to adjust with their environments, and therefore the learning automata of cognitive engines are able to gradually find appropriate actions which results in high accuracy of the proposed cognitive engine (with respect to low *OCD*, *MPL*, and *MPD*). It should be noted that, increasing the value of parameter *z* leading to increasing the required rounds to achieve the lowest value of OCD.

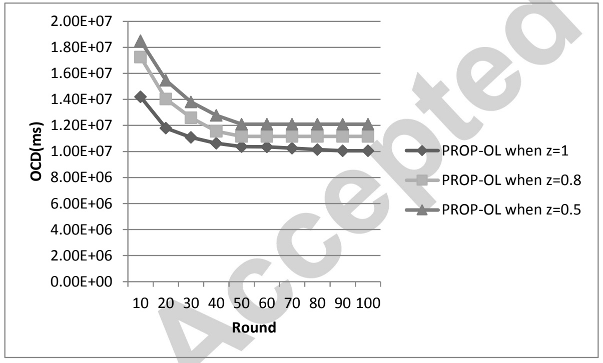

Fig. 25. The impact of parameter *z* on the performance of *PROP-OL* with respect to *OCD* when parameter *t* is 0.6

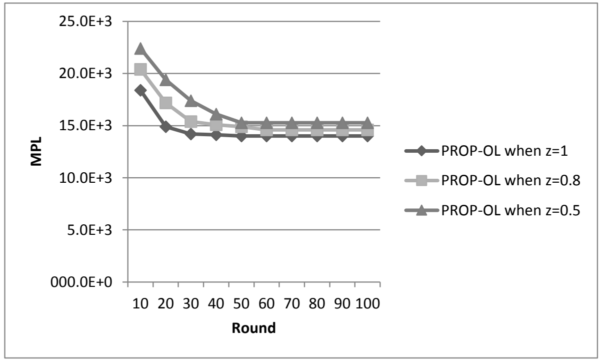

Fig. 26.The impact of parameter *z* on the performance of *PROP-OL* with respect to *MPL* when parameter *t* is 0.6

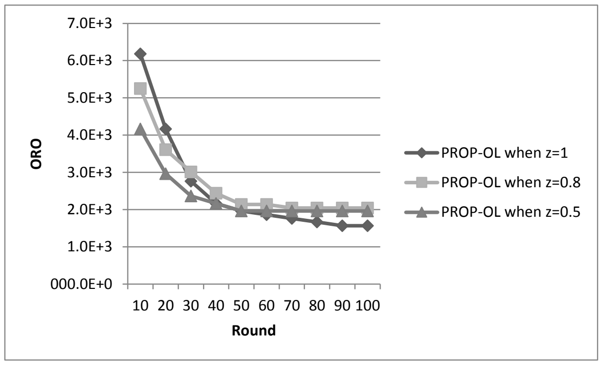

Fig. 27. The impact of parameter *z* on the performance of *PROP-OL* with respect to *ORO* when parameter *t* is 0.6

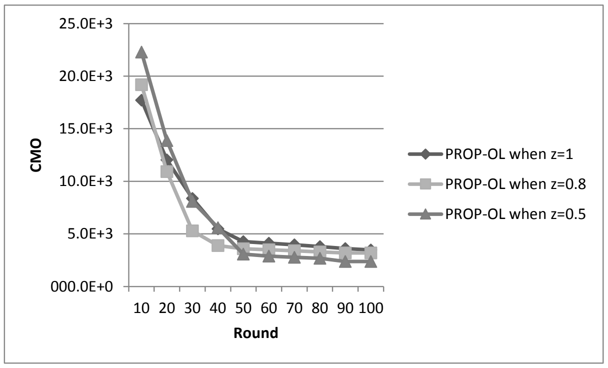

Fig. 28. The impact of parameter *z* on the performance of *PROP-OL* with respect to *CMO* when parameter *t* is 0.6

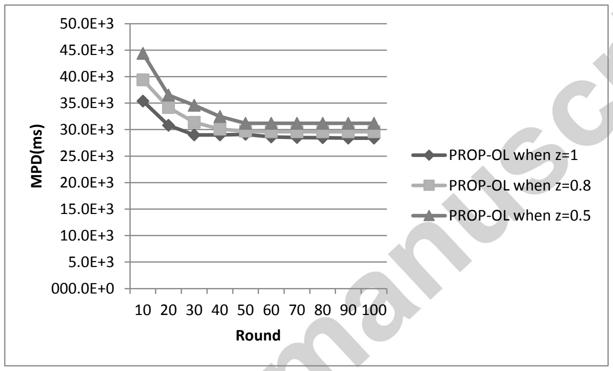

Fig. 29. The impact of parameter *z* on the performance of *PROP-OL* with respect to *MPD* when parameter *t* is 0.6

Table 8. The required number of rounds to the lowest values of OCD for PROP-OL when t=0.6

	PROP-OL when z=0.5	**PROP-OL when z=0.8**	**PROP-OL when z=1**
The required number of rounds	44	54	73
The lowest value of OCD	1.21E+07ms	1.12E+07ms	1.02E+07ms

Experiment 3:

This experiment is conducted to show the impact of the adaptation on the performance of the proposed cognitive engine. For this purpose, the proposed cognitive engine is compared with *X-BOT* and *PROP-O* algorithms which do not use any algorithm to adapt themselves in terms five above mentioned criteria. It should be noted that, some parts of the proposed cognitive engine are borrowed from *PROP-O* algorithm. In the proposed cognitive engine, the value of parameters *z* and *t* are set to 1 and 0.6 respectively. Both of proposed cognitive engine and *PROP-O* algorithms are tested for three initial values for parameter *r* = 1, 2 and 3. *Topology.1* is used as underlay topology. For other topologies similar results have been obtained.

According to the results of this experiment which are shown in Fig. 30 to Fig. 44 and Table 9 we may conclude the following:

- In terms of *OCD*, *MPL*, and *MPD* the proposed cognitive engine performs better than *PROP-O* and *X-BOT* algorithms. This is because of the fact that in the proposed approach, each peer utilizes a cognitive engine for parameter adaptation (parameter neighborhood radius) and adaptive execution of the exchange operator which improves its functionality using the response received from the network as the network operation proceeds. This response which is computed based on some information about the peers and their neighbors reflects the current state of the network and used to improve the behavior of the cognitive engine. Adaptation of neighborhood radius parameter *r* helps each peer to find appropriate *corresponding* and *candidate* peers to be used in the exchange operation leading to lower *OCD*, *MPL*, and *MPD*. In late rounds of the simulation, adaptive execution of exchange operator also helps each peer to better manage the exchange operators in such way that unnecessary exchange operations to be avoided leading to lower *ORO* and *CMO*. In contrast to *X-BOT*, and *PROP-O* algorithms which do not utilize adaptive mechanisms to improve their functionality, the proposed cognitive engine is able to improve the quality of the overlay (with respect to low *OCD*, *MPL,* and *MPD*) and decrease its overhead (with respect to low *ORO* and *CMO*).

- For the proposed cognitive engine the values of *ORO* and *CMO* are higher than other algorithms in early rounds of the simulation. Note that as the time passes the performance of the proposed cognitive engine in terms of *ORO* and *CMO* improves. Note that, that the number of peers which are reconfigured by the proposed cognitive engine (using exchange operator) is decreased at the last round of the simulation. This is because, after finding appropriate neighbors by a peer, the peer is able to decrease its neighborhood radius using its own cognitive engine. Decreasing the neighborhood radius leads to decrease the scope of the local search and decreasing the scope of the local search leads to reconfiguring fewer peers (with respect to low *ORO*) and generating fewer control messages (with respect to low *CMO*). Low performance of the proposed cognitive engine in the early rounds is caused by inappropriate connections of peers in overlay network at the beginning of the operation of the network during which more peers must be reconfigured (using exchange operator) and more control messages must be generated for reconfiguration of the overlay network.

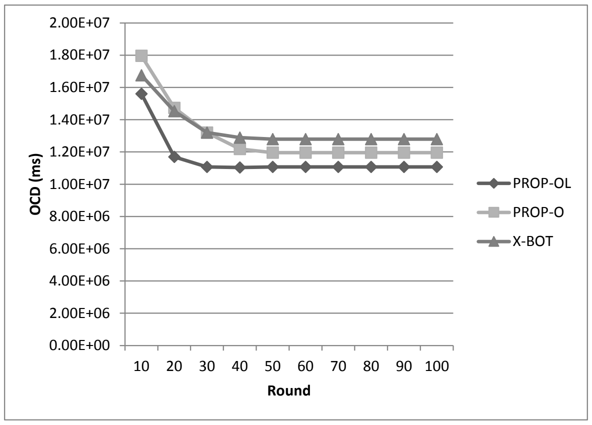

Fig. 30. Comparison of PROP-OL with PROP-O algorithm with respect to OCD when r=1

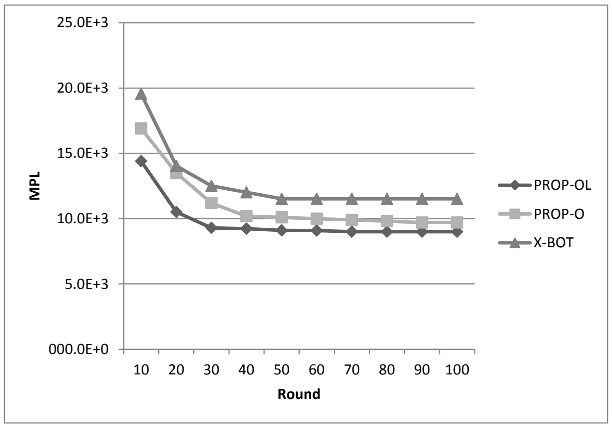

Fig. 31. Comparison of PROP-OL with PROP-O algorithm with respect to MPL when r=1

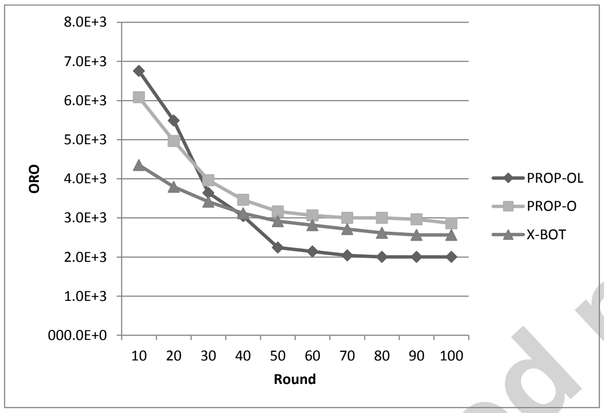

Fig. 32. Comparison of PROP-OL with PROP-O and X-BOT algorithms with respect to ORO when r=1

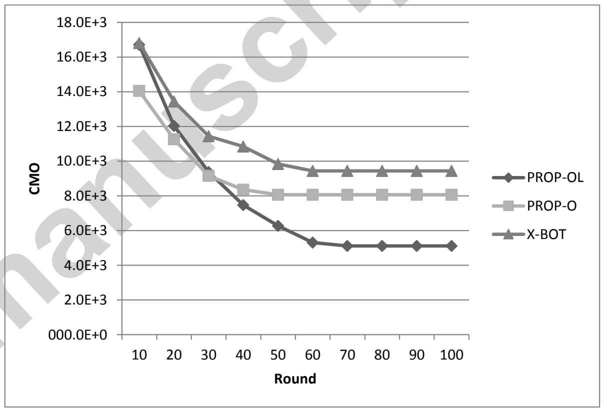

Fig. 33. Comparison of PROP-OL with PROP-O and X-BOT algorithms with respect to CMO when r=1

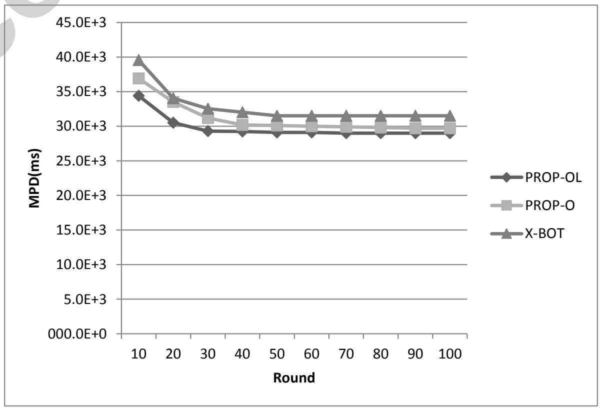

Fig. 34. Comparison of PROP-OL with PROP-O and X-BOT algorithms with respect to MPD when r=1

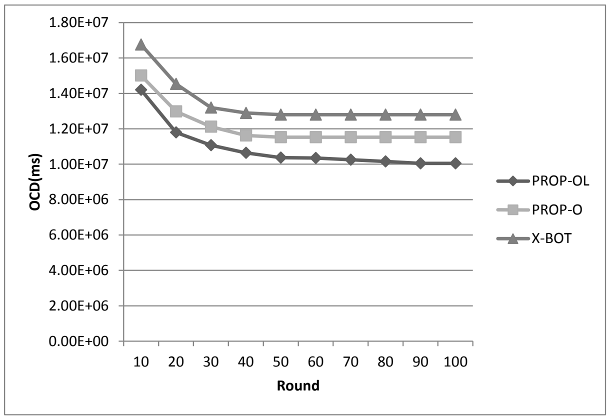

Fig. 35 Comparison of PROP-OL with PROP-O and X-BOT algorithms with respect to OCD when r=2

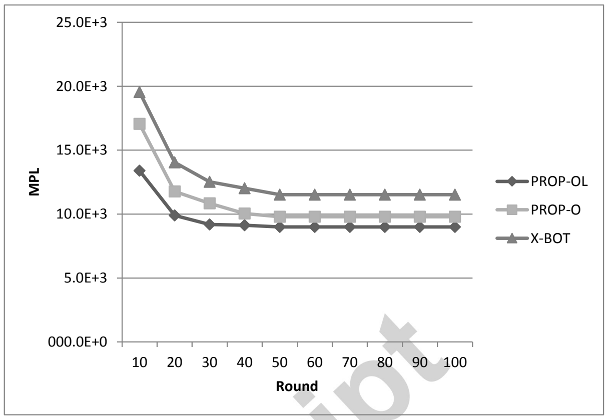

Fig. 36. Comparison of PROP-OL with PROP-O and X-BOT algorithms with respect to MPL when r=2

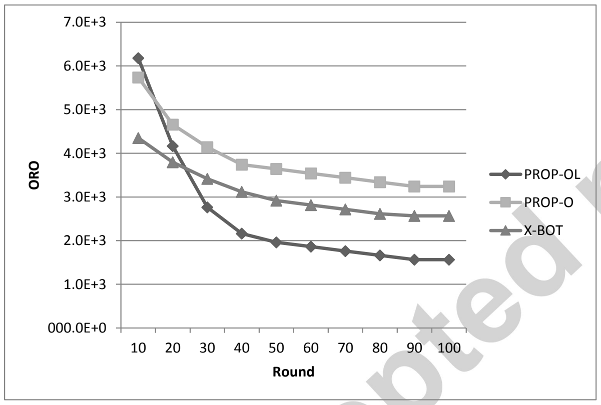

Fig. 37. Comparison of PROP-OL with PROP-O and X-BOT algorithms with respect to ORO when r=2

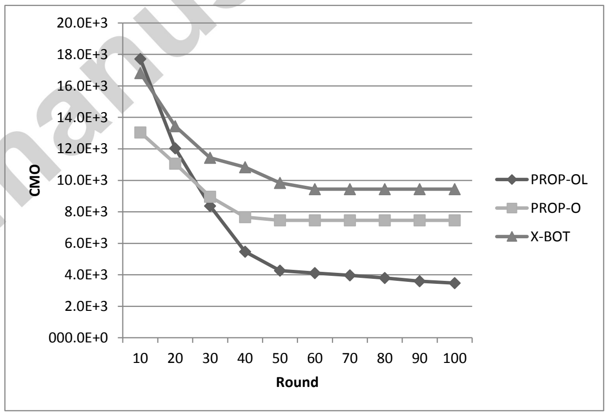

Fig. 38. Comparison of PROP-OL with PROP-O and X-BOT algorithms with respect to CMO when r=2

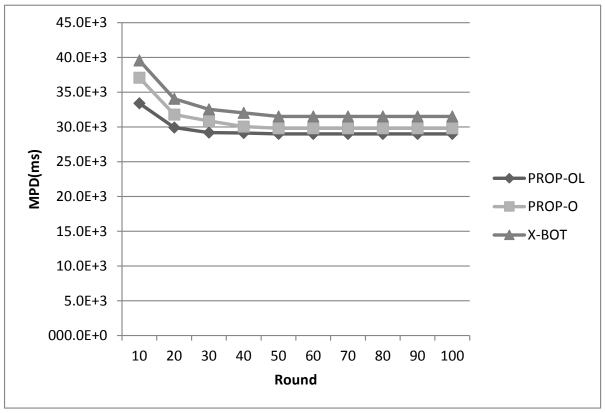

Fig. 39. Comparison of PROP-OL with PROP-O

and X-BOT algorithms with respect to MPD when r=2

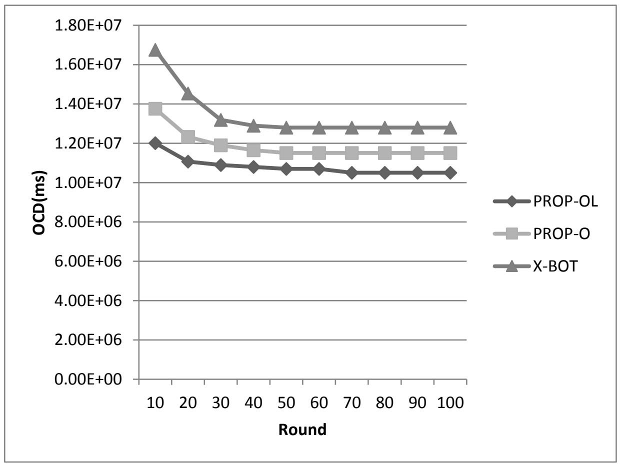

Fig. 40. Comparison of PROP-OL with PROP-O and X-BOT algorithms with respect to OCD when r=3

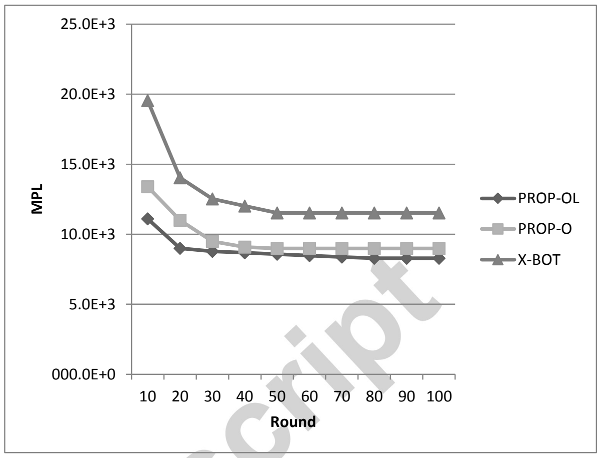

Fig. 41. Comparison of PROP-OL with PROP-O and X-BOT algorithms with respect to MPL when r=3

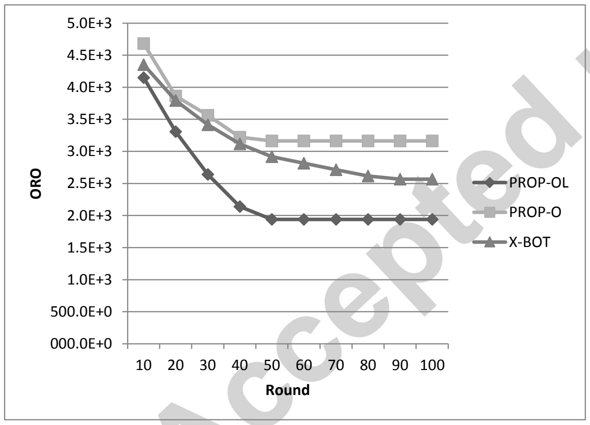

Fig. 42. Comparison of PROP-OL with PROP-O and X-BOT algorithms with respect to ORO when r=3

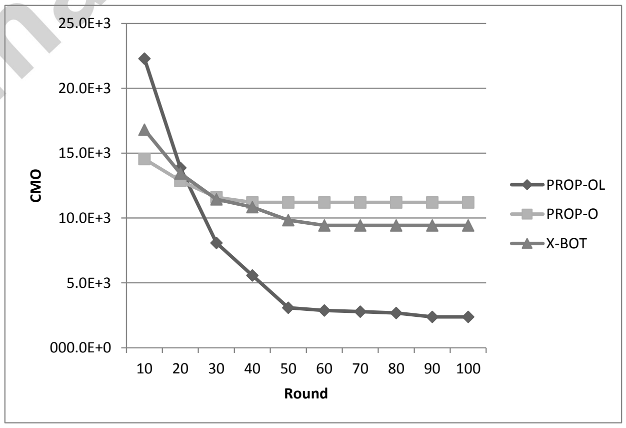

Fig. 43. Comparison of PROP-OL with PROP-O and X-BOT algorithms with respect to CMO when r=3

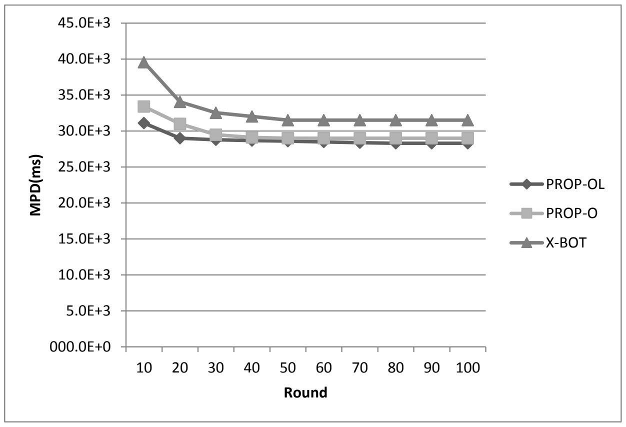

Fig. 44. Comparison of PROP-OL with PROP-O and X-BOT algorithms with respect to MPD when r=3

Table 9. The required number of rounds to the lowest value of OCD

		PROP-OL	**PROP-O**	**X-BOT**
Neighborhood radius r=1	**The required number of rounds**	30	42	45
	The lowest value of OCD	1.11E+07ms	1.20E+07ms	1.28E+07ms
Neighborhood radius r=2	**The required number of rounds**	73	49	45
	The lowest value of OCD	1.02E+07ms	1.15E+07ms	1.28E+07ms
Neighborhood radius r=3	**The required number of rounds**	66	43	45
	The lowest value of OCD	1.05E+07ms	1.15E+07ms	1.28E+07ms

Experiment 4:

In this experiment, we study the performance of the proposed cognitive engine on topologies *Topology.2* and *Topology.3* both belonging to the class of router level topologies [66] with respect to *OCD*, *MPL*, *ORO*,*CMO*, and *MPD*. In the proposed cognitive engine, the value of parameters *z* and *t* are set to 1 and 0.6 respectively. The results are compared with the results obtained for *PROP-O* algorithm when *r*=2 and *X-BOT* algorithm. *Topology.2* contains few and populated groups but *Topology.3* contains many low populated groups in which the distance between the groups is far greater than the distance between the peers in each group. In other word, in *Topology.3* the probability that two peers in the overlay network belong to different autonomous system is high. Fig. 45 to Fig. 49 give the results of this experiment for *Topology.2* and Fig. 50 to Fig. 54 for *Topology3*. From the results we may conclude that for both topologies *Topology2* and *Topology3*, the proposed cognitive engine performs better than *PROP-O* and *X-BOT* in terms of *OCD*, *MPL*, and *MPD*. This means that the proposed cognitive engine can be efficient even for underlay topologies which belonging to the router level topologies. Since the proposed cognitive engine is not dependent to a specific underlay network it can be useful for different types of the underlay networks.

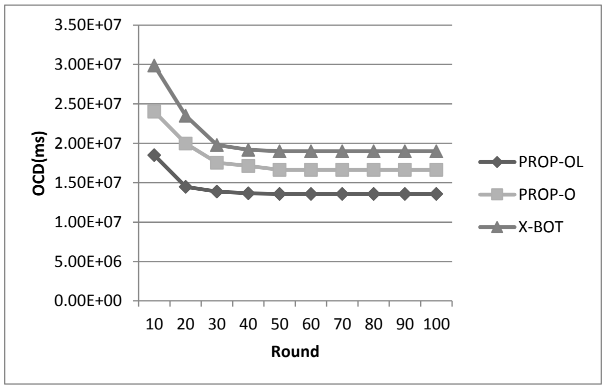

Fig. 45. Comparison of PROP-OL with PROP-O and X-BOT algorithms with respect to OCD when Topology.2 is used

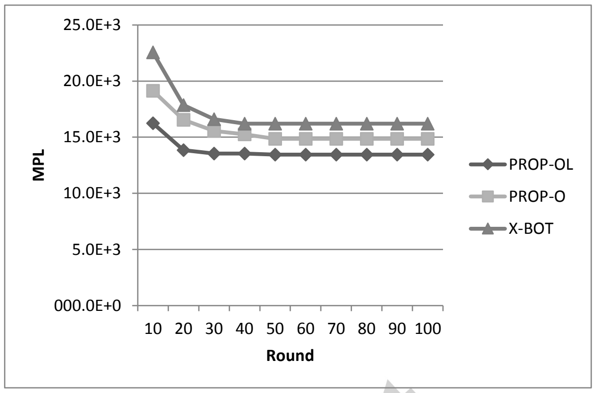

Fig. 46. Comparison of PROP-OL with PROP-O and X-BOT algorithms with respect to MPL when Topology.2 is used

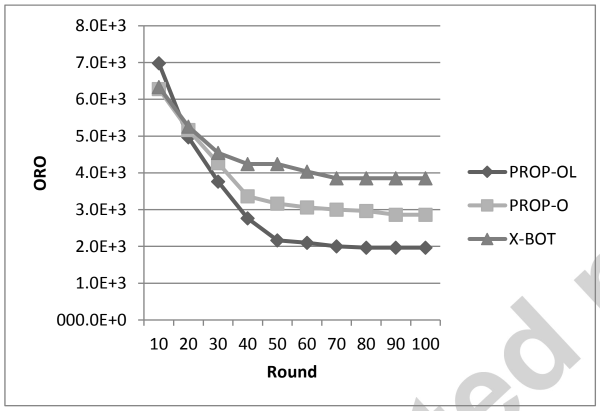

Fig. 47. Comparison of PROP-OL with PROP-O and X-BOT algorithms with respect to ORO when Topology.2 is used

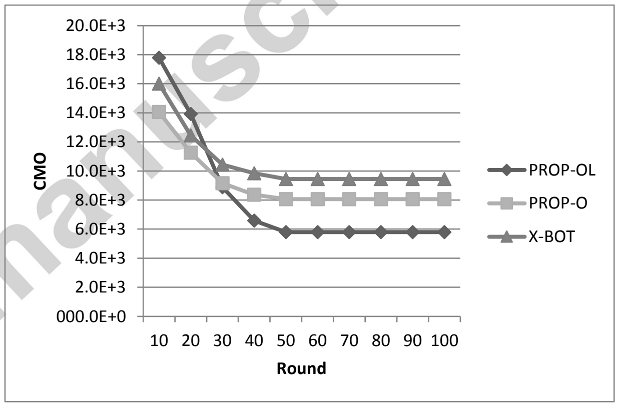

Fig. 48. Comparison of PROP-OL with PROP-O and X-BOT algorithms with respect to CMO when Topology.2 is used

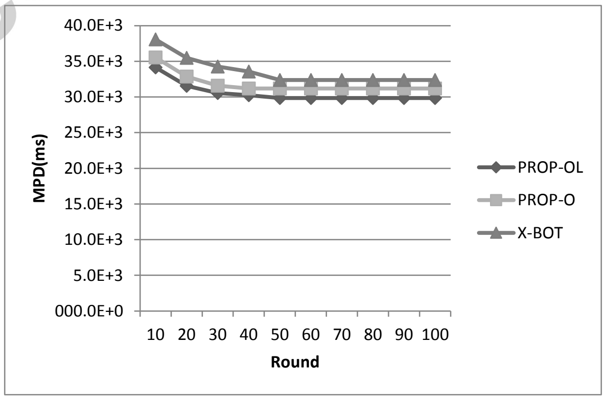

Fig. 49. Comparison of PROP-OL with PROP-O and X-BOT algorithms with respect to MPD when Topology.2 is used

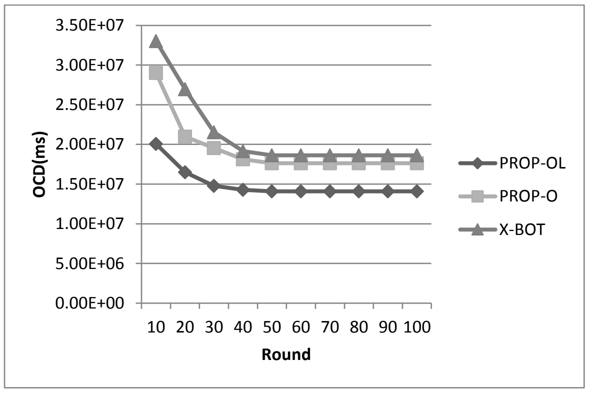

Fig. 50. Comparison of PROP-OL with PROP-O and X-BOT algorithms with respect to OCD when Topology.3 is used

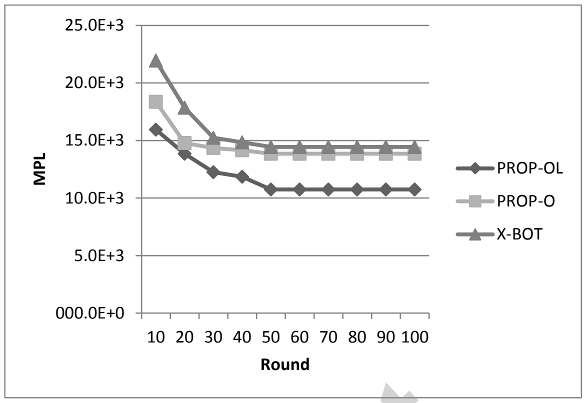

Fig. 51. Comparison of PROP-OL with PROP-O and X-BOT algorithms with respect to MPL when Topology.3 is used

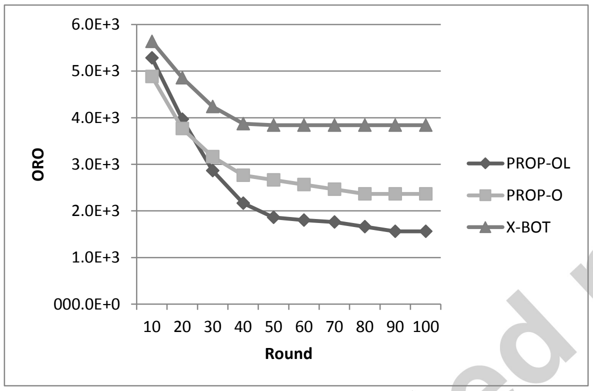

Fig. 52. Comparison of PROP-OL with PROP-O and X-BOT algorithms with respect to ORO when Topology.3 is used

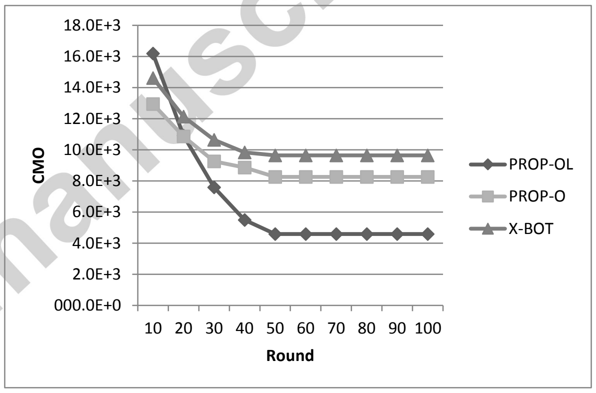

Fig. 53. Comparison of PROP-OL with PROP-O and X-BOT algorithms with respect to CMO when Topology.3 is used

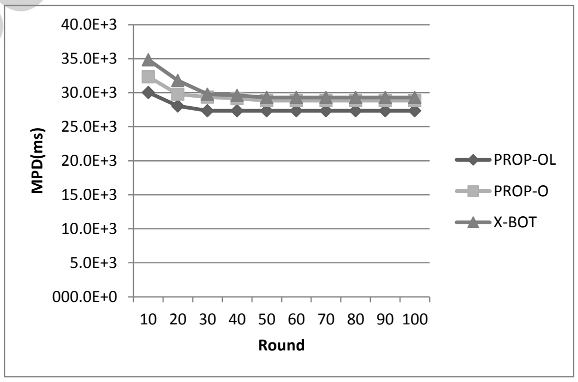

Fig. 54. Comparison of PROP-OL with PROP-O and X-BOT algorithms with respect to MPD when Topology.3 is used

Table 10. The required number of rounds to the lowest value of OCD

		PROP-OL	PROP-O	X-BOT
Topology.2	**The required number of rounds**	48	42	44
	The lowest value of OCD	1.37E+07ms	1.66E+07ms	1.90E+07ms
Topology.3	**The required number of rounds**	50	45	42
	The lowest value of OCD	1.43E+07ms	1.76E+07ms	1.86E+07ms

Experiment 5:

This experiment is conducted to study the impact of different churn models on the performance of the proposed cognitive engine when *r*=2 and *Topology.1* is used as underlay topology. In the proposed cognitive engine, the value of parameters *z* and *t* are set to 1 and 0.6 respectively. The used churn models are Random churn model [65] and Pareto churn model [65] according to Table 3. In Table 3, the probabilities of joining and leaving the peers for one Random churn models and the lifetime mean and the dead time mean for one Pareto churn model used in this experiment are given. The results obtained from the proposed cognitive engine are compared with the results for *PROP-O* and *X-BOT* algorithms. Fig. 55 to Fig. 64 give the results of this experiment. From the results we may conclude the following:

- In terms of *OCD*, *MPL*, and *MPD*, the proposed cognitive engine performs better than *PROP-O* and *X-BOT* algorithms for all churn models. This is because of the fact that the proposed cognitive engine tune up itself in a self-adaptive manner which leads to finding better *candidate* peers (for exchange operation) and therefore results in fewer number of mismatched paths which leading to low *MPL* and *MPD*, and also low *OCD*.

- In terms of *ORO* and *CMO* the proposed cognitive engine performs worse than *PROP-O* and *X-BOT* algorithms at early rounds of the simulation for all the churn models. This is because each peer cooperates with its neighbors to improve the decisions of its cognitive engine during the operation of the network. Because of cooperation among peers, the information about the joining and leaving peers will be propagated in the network. The information propagation among peers increase the amount of control messages and motivates the cognitive engines of more peers to consider the changes were made in the network which leads to increasing the changes of the configuration of the peers (with respect to high *ORO* and high *CMO*).

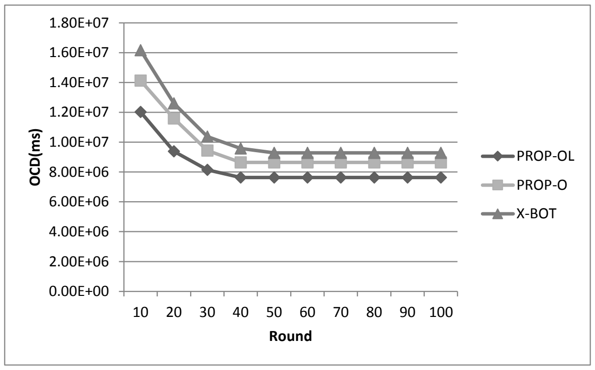

Fig. 55. Comparison of PROP-OL with PROP-O and X-BOT algorithms with respect to OCD when Random churn is used

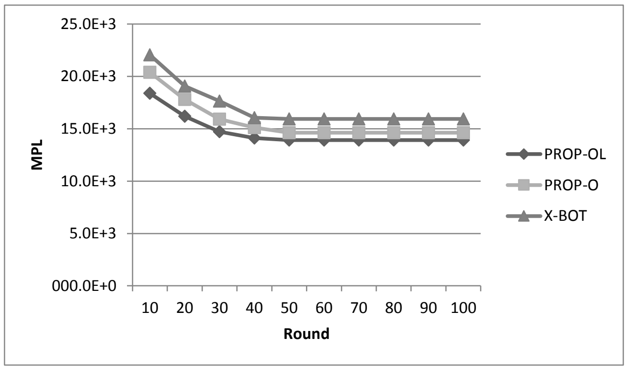

Fig. 56. Comparison of PROP-OL with PROP-O and X-BOT algorithms with respect to MPL when Random churn is used

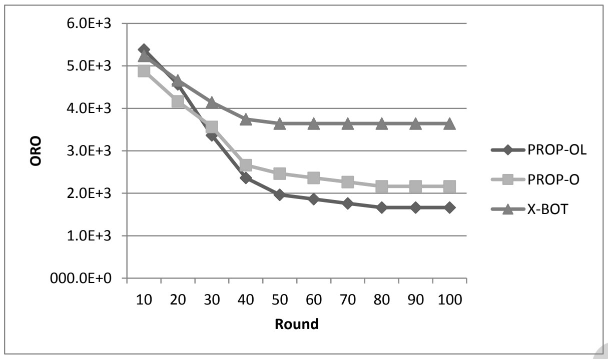

Fig. 57. Comparison of PROP-OL with PROP-O and X-BOT algorithms with respect to ORO when Random churn is used

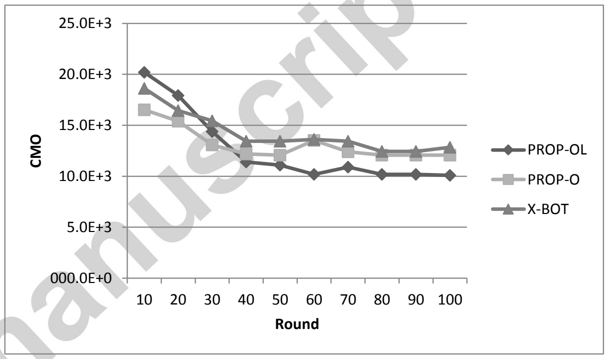

Fig. 58. Comparison of PROP-OL with PROP-O and X-BOT algorithms with respect to CMO when Random churn is used

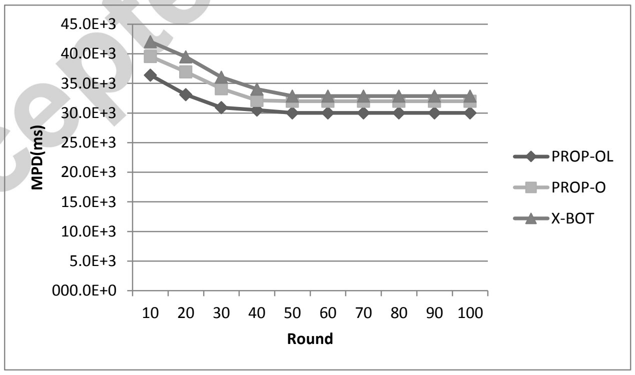

Fig. 59. Comparison of PROP-OL with PROP-O and X-BOT algorithms with respect to MPD when Random churn is used

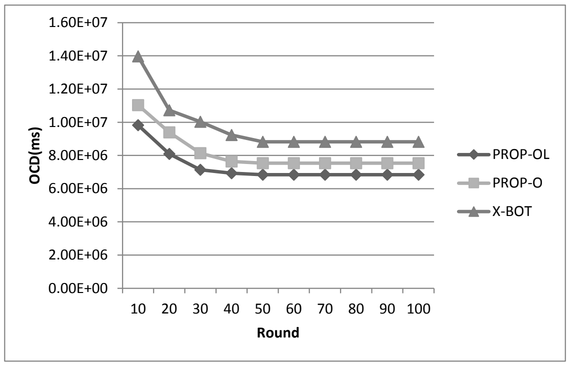

Fig. 60. Comparison of PROP-OL with PROP-O and X-BOT algorithms with respect to OCD when Pareto Churn is used

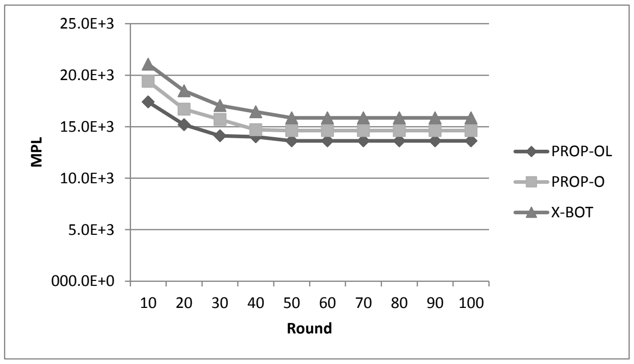

Fig. 61. Comparison of PROP-OL with PROP-O and X-BOT algorithms with respect to MPL when Pareto Churn is used

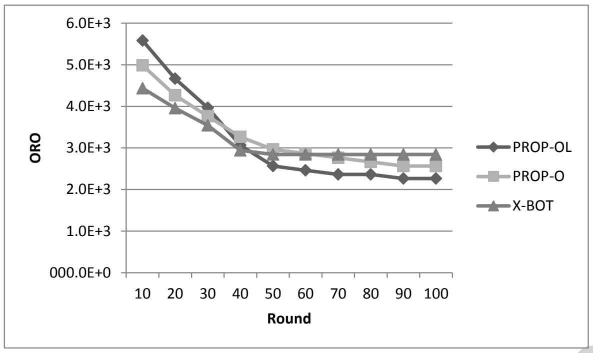

Fig. 62. Comparison of PROP-OL with PROP-O and X-BOT algorithms with respect to ORO when Pareto Churn is used

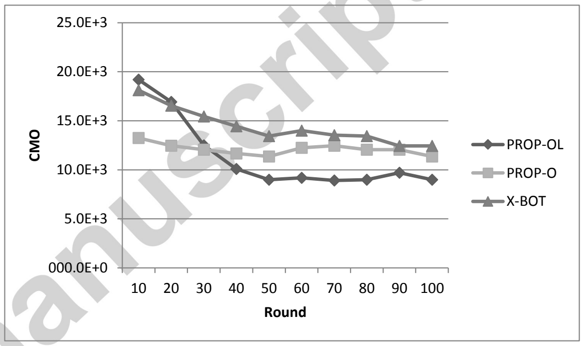

Fig. 63. Comparison of PROP-OL with PROP-O and X-BOT algorithms with respect to CMO when Pareto Churn is used

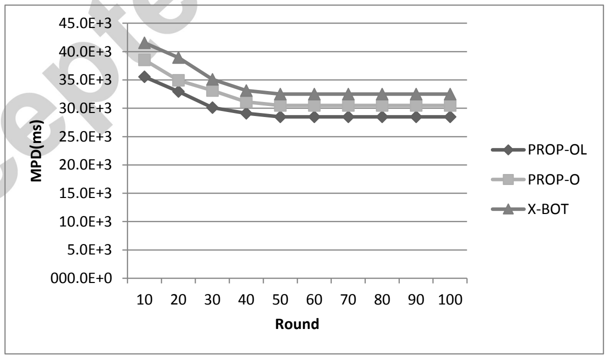

Fig. 64. Comparison of PROP-OL with PROP-O and X-BOT algorithms with respect to MPD when Pareto Churn is used

Table 11. The required number of rounds to the lowest value of OCD

		PROP-OL	**PROP-O**	**X-BOT**
Random Churn	**The required number of rounds**	47	38	44
	The lowest value of OCD	7.63E+06ms	8.64E+06ms	9.28E+06ms
Pareto Churn	**The required number of rounds**	46	41	45
	The lowest value of OCD	6.84E+06ms	7.54E+06ms	8.83E+06ms

6 Conclusion

In this paper, a framework for cognitive peer-to-peer networks was introduced and then an approach based on cellular learning automata for designing cognitive engines in the cognitive peer-to-peer networks was proposed. The proposed approach was used to suggest a cognitive engine called *PROP-OL* for solving topology mismatch problem. In the proposed approach, each peer in the network, in interaction with its neighboring peers adaptively changes its configuration using its cognitive engine during the operation of the network. To evaluate the suggested cognitive engine several computer simulations have conducted in *OverSim*. To show the superiority of the suggested cognitive engine, it is compared with *PROP-O* and *X-BOT* algorithms. Experimental results showed that the suggested cognitive engine can compete with *PROP-O* and *X-BOT* algorithms with respect to end-to-end delays and delays of mismatched paths. In the future, we plan to use the proposed framework in hybrid architectures such as peer-to-peer clouds and peer-to-peer social networks.

Acknowledgements

This paper is dedicated to my late mother Dr. H. Afsar Lajevardi. I will never forget her kindness and support. The authors would like to thank the editor and anonymous reviewers whose helpful comments improved the quality of this paper.

References

[1] Q. H. Mahmoud, *Cognitive networks*. Wiley Online Library, 2007.

[2] J. Mitola, “Cognitive radio: an integrated agent architecture for software defined radio,” PhD dissertation, Royal Institute of Technology (KTH), Stockholm, Sweden, 2000.

[3] J. Mitola and G. Q. Maguire Jr, “Cognitive radio: making software radios more personal,” *IEEE Personal Communications*, vol. 6, no. 4, pp. 13–18, 1999.

[4] S. Haykin, “Cognitive radio: brain-empowered wireless communications,” *IEEE Journal on Selected Areas in Communications*, vol. 23, no. 2, pp. 201–220, 2005.

[5] R. Dipankar, N. B. Mandayam, J. B. Evans, B. J. Ewy, S. Seshan, and P. Steenkiste, “CogNet: an architectural foundation for experimental cognitive radio networks within the future internet.,” in *Proceedings of first ACM/IEEE international workshop on Mobility in the evolving internet architecture*, San Francisco, California, USA, 2006, pp. 11–16.

[6] H. Jian, P. Jun, J. Fu, Q. Gaorong, and L. Weirong, “A Distributed Q Learning Spectrum Decision Scheme for Cognitive Radio Sensor Network,” *Hindawi Publishing Corporation*, no. 2015, pp. 1550–1329, 2015.

[7] Y. Song, C. Zhang, and Y. Fang, “Stochastic traffic engineering in multihop cognitive wireless mesh networks,” *IEEE Transactions on Mobile Computing*, vol. 9, no. 3, pp. 305–316, 2010.

[8] G. Sakellari, “The cognitive packet network: a survey,” *The Computer Journal*, vol. 53, no. 3, pp. 268–279, 2010.

[9] Y. Wu and I. Niemegeers, “A cognitive architecture for personal networks,” in *Autonomic Networking*, Paris, France, 2006, pp. 12–24.

[10] R. W. Thomas, D. H. Friend, L. A. DaSilva, and A. B. MacKenzie, “Cognitive networks: adaptation and learning to achieve end-to-end performance objectives,” *IEEE Communications Magazine*, vol. 44, no. 12, pp. 51–57, 2007.

[11] R. W. Thomas, L. A. DaSilva, and A. B. MacKenzie, “Cognitive networks,” in *First IEEE International Symposium on New Frontiers in Dynamic Spectrum Access Networks*, Baltimore, MD, USA, 2005, pp. 352–360.

[12] R. Thomas, “Cognitive Networks,” PhD dissertation, Virginia Tech, Blacksburg, Virginia, 2007.

[13] E. Gelenbe, R. Lent, and Z. Xu, “Design and performance of cognitive packet networks* 1,” *Performance Evaluation*, vol. 46, no. 2–3, pp. 155–176, 2001.

[14] E. Gelenbe, “A software defined self-aware network: The cognitive packet network,” in *IEEE 3rd Symposium on Network Cloud Computing and Applications*, Rome, 2014, pp. 9–14.

[15] A. Galindo-Serrano and L. Giupponi, “Distributed Q-learning for aggregated interference control in cognitive radio networks,” *IEEE Transactions on Vehicular Technology*, vol. 59, no. 4, pp. 1823–1834, 2010.

[16] M. Lee, D. Marconett, X. Ye, and S. J. Yoo, “Cognitive network management with reinforcement learning for wireless mesh networks,” in *Proceedings of the 7th IEEE international conference on IP operations and management,* San Jos, USA, 2007, pp. 168–179.

[17] Z. Song, B. Shen, Z. Zhou, and K. S. Kwak, “Improved ant routing algorithm in cognitive radio networks,” in *9th International Symposium on Communications and Information Technology*, Incheon, Korea, 2009, pp. 110–114.

[18] Q. He, Z. Feng, and P. Zhang, “Reconfiguration Decision Making Based on Ant Colony Optimization in Cognitive Radio Network,” *Wireless Personal Communications*, vol. 71, no. 2, pp. 1–23, 2012.

[19] D. H. Friend, M. Y. ElNainay, Y. Shi, and A. B. MacKenzie, “Architecture and performance of an island genetic algorithm-based cognitive network,” in *Consumer Communications and Networking Conference*, Las Vegas, NV, 2008, pp. 993–997.

[20] T. W. Rondeau, B. Le, C. J. Rieser, and C. W. Bostian, “Cognitive radios with genetic algorithms: Intelligent control of software defined radios,” in *Software defined radio forum technical conference*, Phoenix, Arizona, USA, 2004, pp. 13–18.

[21] Y. K. Kwok, *Peer-to-Peer Computing: Applications, Architecture, Protocols, and Challenges*. United States: CRC Press, 2011.

[22] R. Rajiv and L. Zhao, “Peer-to-peer service provisioning in cloud computing environments,” *The Journal of Supercomputing*, vol. 63, no. 1, pp. 154–184, 2013.

[23] M. Amoretti, A. Graziolo, and F. Zanichelli, “An autonomic approach for P2P/cloud collaborative environments,” *Peer-to-Peer Networking and Applications*, pp. 1–16, 2015.

[24] K. Ryota and S. Fujita, “Peer-to-Peer Content Delivery System with Bounded Traffic between Autonomous Systems,” presented at the First International Symposium on Computing and Networking (CANDAR), Matsuyama, 2013, pp. 630–632.

[25] M. Garmehi, M. Analoui, M. Pathan, and R. Buyya, “An economic replica placement mechanism for streaming content distribution in Hybrid CDN-P2P networks,” *Computer Communications*, vol. 52, pp. 60–70, 2014.

[26] M. Garmehi, M. Analoui, M. Pathan, and R. Buyya, “An economic mechanism for request routing and resource allocation in hybrid CDN–P2P networks,” *International Journal of Network Management*, 2015.

[27] H. Kim and S. Lee, “A Peer to Peer Social Networking Service Exploiting Triangle Relationship among Friends,” in *Fifth International Conference on Ubiquitous and Future Networks*, Da Nang, 2013, pp. 816–821.

[28] T. Kim and S. Lee, “Efficient social graph augmentation schemes for a peer to peer social networking service,” in *International Conference on Big Data and Smart Computing*, Bangkok, 2014, pp. 269–270.

[29] O. Babaoglu, H. Meling, and A. Montresor, “Anthill: a framework for the development of agent-based peer-to-peer systems,” in *22nd International Conference on Distributed Computing Systems*, Vienna, Austria, 2002, pp. 15–22.

[30] A. Forestiero, C. Mastroianni, and M. Meo, “Self-Chord: A bio-inspired algorithm for structured P2P systems,” in *IEEE/ACM International Symposium on Cluster Computing and the Grid*, Shanghai,China, 2009, pp. 44–51.

[31] P. L. Snyder, R. Greenstadt, and G. Valetto, “Myconet: A fungi-inspired model for superpeer-based peer-to-peer overlay topologies,” in *Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, San Francisco, CA, 2009, pp. 40–50.

[32] M. Dumitrescu and R. Andonie, “Clustering superpeers in p2p networks by growing neural gas,” in *20th Euromicro International Conference on Parallel, Distributed and Network-Based Processing*, Munich, Germany, 2012, pp. 311–318.

[33] N. Ganguly and 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.

[34] F. Sharifkhani and M. R. Pakravan, “Bacterial foraging search in unstructured P2P networks,” in *27th Canadian Conference on Electrical and Computer Engineering*, Toronto, ON, 2014, pp. 1–8.

[35] A. Singh and M. Haahr, “Creating an adaptive network of hubs using Schelling’s model,” *Communications of the ACM*, vol. 49, no. 3, pp. 69–73, 2006.

[36] K. Sripanidkulchai, “A measurement-driven approach to designing peer-to-peer systems,” Doctoral Dissertation, Carnegie Mellon University, Pittsburgh, PA, USA, 2005.

[37] N. TS Eugene, Y. Chu, S. G.Rao, K. Sripanidkulchai, and H. Zhang, “Measurement-Based Optimization Techniques for Bandwidth-Demanding Peer-to-Peer Systems,” in *Twenty-Second Annual Joint Conference of the IEEE Computer and Communications*, San Franciso, CA, USA, 2003, vol. 3, pp. 2199 – 2209.

[38] K. Sripanidkulchai, B. Maggs, and H. Zhang, “Efficient content location using interest-based locality in peer-to-peer systems,” in *Twenty-Second Annual Joint Conference of the IEEE Computer and Communications*, San Francisco, California, USA, 2003, vol. 3, pp. 2166–2176.

[39] H. Luan, K.-W. Kwong, X. Hei, and D. H.K.Tsang, “Adaptive topology formation for peer-to-peer video streaming,” *Peer-to-peer Networking and Applications*, vol. 3, no. 3, pp. 186–207, 2010.

[40] T. Qiu, E. Chan, M. Ye, G. Chen, and B. Y. Zhao, “Peer-exchange schemes to handle mismatch in peer-to-peer systems,” *The Journal of Supercomputing*, vol. 48, no. 1, pp. 15–42, 2009.

[41] Y. Liu, “A two-hop solution to solving topology mismatch,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, pp. 1591–1600, 2008.

[42] H. Rostami and J. Habibi, “Topology awareness of overlay P2P networks,” *Concurrency and Computation: Practice and Experience*, vol. 19, no. 7, pp. 999–1021, 2007.

[43] R. W. Thomas, D. H. Friend, L. A. DaSilva, and A. B. MacKenzie, “Cognitive Networks,” in *Cognitive Radio, Software Defined Radio, and Adaptive Wireless Systems*, vol. 2, Springer Netherlands, 2007, pp. 17–41.

[44] T. C. Schelling, “Dynamic models of segregation†,” *Journal of mathematical sociology*, vol. 1, no. 2, pp. 143–186, 1971.

[45] N. G. Domic, E. Goles, and S. Rica, “Dynamics and complexity of the schelling segregation model,” *Physical Review E*, vol. 83, no. 5, pp. 96–111, 2011.

[46] J. Leitão, J. P. Marques, J. Pereira, and L. Rodrigues, “X-bot: A protocol for resilient optimization of unstructured overlay networks,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 11, pp. 2175–2188, 2012.

[47] S. Wolfram, “Theory and applications of cellular automata,” *World Scientific Publication*, 1986.

[48] C. Somarakis, G. Papavassilopoulos, and F. Udwadia, “A dynamic rule in cellular automata,” in *22nd European Conference on Modelling and Simulation*, Nicosia, Cyprus, 2008, pp. 164–170.

[49] S. Dantchev, “Dynamic Neighbourhood Cellular Automata,” *Computer Journal*, vol. 54, no. 1, pp. 26–32, 2011.

[50] A. Ilachinski and P. Halpern, “Structurally dynamic cellular automata,” *Complex Systems*, vol. 1, no. 3, pp. 503–527, 1987.

[51] K. S. Narendra and M. A. L. Thathachar, *Learning Automata: An Introduction*. Englewood Cliffs, NJ: Prentice-Hall, 1989.

[52] M. Thathachar and P. S. Sastry, *Networks of Learning Automata: Techniques for Online Stochastic Optimization*. Dordrecht, Netherlands: Kluwer Academic Publishers, 2004.

[53] H. Beigy and M. R. Meybodi, “A mathematical framework for cellular learning automata,” *Advances in Complex Systems*, vol. 3, no. 4, pp. 295–319, 2004.

[54] H. Beigy and M. R. Meybodi, “Open synchronous cellular learning automata,” *Advances in Complex Systems*, vol. 10, no. 4, pp. 527–556, 2007.

[55] H. Beigy and M. R. Meybodi, “Asynchronous cellular learning automata,” *Automatica*, vol. 44, no. 5, pp. 1350–1357, 2008.

[56] M. Esnaashari and M. R. Meybodi, “Irregular Cellular Learning Automata,” *IEEE Transactions on Cybernetics*, no. 99, p. 1, 2014.

[57] H. Beigy and M. R. Meybodi, “Cellular learning automata with multiple learning automata in each cell and its applications,” *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 40, no. 1, pp. 54–65, 2010.

[58] M. Esnaashari and M. Meybodi, “Deployment of a Mobile Wireless Sensor Network with k-Coverage Constraint: A Cellular Learning Automata Approach,” *Wireless Networks*, vol. 19, no. 5, pp. 945–968, 2013.

[59] M. Esnaashari and M. Meybodi, “A cellular learning automata-based deployment strategy for mobile wireless sensor networks,” *Journal of Parallel and Distributed Computing*, vol. 71, no. 5, pp. 988–1001, 2011.
[60] M. Esnaashari and M. R. Meybodi, “Dynamic point coverage problem in wireless sensor networks: a cellular learning automata approach,” *Ad Hoc & Sensor Wireless Networks*, vol. 10, no. 2–3, pp. 193–234, 2010.
[61] M. Esnaashari and M. Meybodi, “A cellular learning automata based clustering algorithm for wireless sensor networks,” *Sensor Letters*, vol. 6, no. 5, pp. 723–735, 2008.
[62] Y. Zhao, W. Jiang, S. Li, Y. Ma, G. Su, and X. Lin, “A cellular learning automata based algorithm for detecting community structure in complex networks,” *Neurocomputing*, vol. 151, pp. 1216–1226, 2015.
[63] R. Rastegar, M. R. Meybodi, and A. Hariri, “A new fine-grained evolutionary algorithm based on cellular learning automata,” *International Journal of Hybrid Intelligent Systems*, vol. 3, no. 2, pp. 83–98, 2006.
[64] V. Moustakas, H. Akcan, M. Roussopoulos, and A. Delis, “Alleviating the topology mismatch problem in distributed overlay networks : A survey,” *The Journal of Systems and Software*, no. 113, pp. 216–245, 2016.
[65] I. Baumgart, B. Heep, and 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.
[66] L. Li, D. Alderson, W. Willinger, and J. Doyle, “A first-principles approach to understanding the internet’s router-level topology,” *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 4, pp. 3–14, 2004.
[67] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, “Making gnutella-like p2p systems scalable,” in *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, Karlsruhe, Germany, 2003, pp. 407–418.
[68] “SNAP: Network datasets: Gnutella peer-to-peer network,” *SNAP: Network datasets: Gnutella peer-to-peer network*, 2014. [Online]. Available: http://snap.stanford.edu/data/p2p-Gnutella04.html. [Accessed: 23-Jan-2014].
[69] P. Mahadevan, D. Krioukov, M. Fomenkov, B. Huffaker, X. Dimitropoulos, and A. Vahdat, “Lessons from three views of the internet topology,” University of California, San Diego, CA, USA, tr-2005-02, 2005.
[70] B. Huffaker, D. Plummer, D. Moore, and 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.

Highlights

- This paper presents a new framework for cognitive peer-to-peer networks.
- We proposed a new approach based on cellular learning automata for designing cognitive engines in cognitive peer-to-peer networks.
- We proposed a cognitive engine for solving topology mismatch problem in unstructured peer-to-peer networks.