J Netw Syst Manage
DOI 10.1007/s10922-015-9353-9

A Self-adaptive Algorithm for Topology Matching in Unstructured Peer-to-Peer Networks

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

Received: 10 May 2014 / Accepted: 8 September 2015

Abstract 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 a 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. In some of the topology matching algorithms called the heuristic algorithms, 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 suffer from two problems; neither the neighborhood radius nor the local operator can adapt themselves to the dynamicity of the network. In this paper, a topology matching algorithm that uses *learning automata* to adapt the neighborhood radius and an adaptation mechanism inspired from the *Schelling segregation model* to manage the execution of the local operator is proposed. To evaluate the proposed algorithm, computer simulations were conducted and then the results were compared with the results obtained for other existing algorithms. Simulation results have shown that the proposed algorithm outperforms the existing algorithms with respect to end-to-end delay and number of mismatched paths.

Keywords Mismatched paths · Local search · Local operator · Schelling segregation model · Learning automata

✉ Mohammad Reza Meybodi
mmeybodi@aut.ac.ir

Ali Mohammad Saghiri
a_m_saghiri@aut.ac.ir

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

Published online: 05 October 2015

1 Introduction

Peer-to-peer networks are computer networks in which all peers have the same role (client and server). In these networks, all peers communicate directly with each other. These networks are overlay networks that are constructed over underlay networks. There are two different types for peer-to-peer networks: structured and unstructured. In unstructured peer-to-peer networks such as *Gnutella* [1], *GIA* [2], and *Freenet* [3], there are some lightweight algorithms to manage the overlay topology. Unstructured peer-to-peer networks are widely used because their design are simple. In structured peer-to-peer networks such as *Chord* [4], and *CAN* [5], distributed algorithms are provided to manage the overlay topology that can provide efficient resource locating algorithms. In structured peer-to-peer networks, any resource can be located within a bounded number of hops. A major consequence of the network dynamicity caused by joining and leaving peers in both structured and unstructured peer-to-peer networks is that the overlay topology can change drastically over time. The network changes might adversely affect the performance of the overlay topology in terms of traffic and end-to-end delay. This issue is more important in an unstructured peer-to-peer network because in such a network, peer dynamics can rapidly change an initially good topology to an ineffective one. To prevent the performance degradations caused by the dynamicity of peer-to-peer networks, the topology management algorithms should be able to adapt themselves to the dynamic conditions of the network [1]. In unstructured peer-to-peer networks, for designing the topology management algorithms several problems such as topology mismatching [6–9] and topology clustering [10] must be solved. Since the primary goal of this paper is to design an adaptive algorithm for topology mismatch problems in peer-to-peer networks, the rest of the introduction section will be devoted to a more detailed discussion about this problem.

Consider n peers that are connected to each other through an overlay network over an underlay network. The topology of the overlay network can be represented by an overlay graph $G = (V, E)$ where $V = \{peer_1, peer_2, \ldots, peer_n\}$ is a set of peers and $E \subseteq V \times V$ is a set of links connecting the peers. The topology of the underlay network can be also represented by an underlay graph $G' = (V', E')$ where

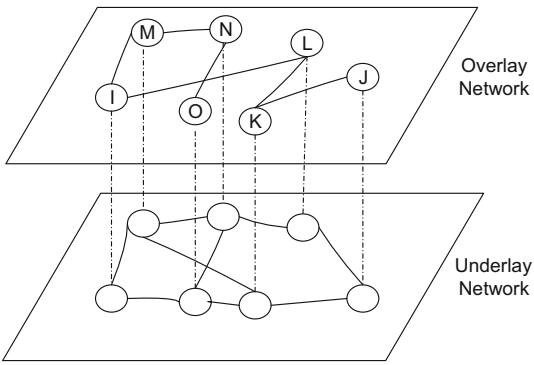

Fig. 1 An example of topology mismatch problem

$V' = \{position_1, position_2, \ldots, position_n\}$ is the set of positions in the underlay network and $E' \subseteq V' \times V'$ is a set of links connecting the positions. In the peer-to-peer network each peer in V is mapped to a position in V' according to a one-to-one function $H : V \rightarrow V'$. In both structured peer-to-peer networks and unstructured peer-to-peer networks, peers choose their neighbors without considering underlay positions and for this reason, the topology of the overlay network may have been mismatched with the topology of the underlay network. Creating a large number of mismatched paths is one of the problems that topology mismatching causes. Figure 1 shows an example of a mismatched path. To transfer a message from L to J ($L \rightarrow J$) in the overlay network path $L \rightarrow K \rightarrow J$ is used, but in the underlay network, we may have an inefficient path such as ($L \rightarrow J \rightarrow K \rightarrow J$). 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 ($L \rightarrow J$) is called a mismatched path. Since the mismatched paths cause redundant end-to-end delays, a type of topology matching algorithm, such as those reported in [6, 7, 11, 12] try to reconfigure the overlay network using local operators in order to decrease redundant delays caused by the mismatched paths. Therefore, the sum of end-to-end delays of the overlay network computed using (1) is used as an objective function that should be minimized by the matching algorithms.

$$M(G,H) = \sum_{peer_i \in V} \sum_{peer_j \in V - \{peer_i\}} I(E,(i,j)) \times d_{ij} \tag{1}$$

In (1), $d_{ij} = D(H(peer_i), H(peer_j))$ is the end-to-end delay from $peer_i$ to $peer_j$, function $D : V' \times V' \rightarrow \mathbb{R}$ computes the end-to-end delay between the pairs of positions in the underlay network, function $H : V \rightarrow V'$ maps a peer in the overlay network into a position in the underlay network and function I is an indicator function that takes a set of links E and a link (i, j) and returns 1 if link (i, j) is in E and 0 otherwise.

In some of the topology matching algorithms, each peer uses a local search operator for gathering information about the neighbors of that peer located in its neighborhood radius (in the overlay network). In these algorithms each peer also uses a local operator for changing the connections among the peers with the aim of decreasing the number of mismatched paths of the overlay. This type of algorithm is used in both structured [13, 14, 15] and unstructured peer-to-peer networks [6, 7, 16]. These algorithms reconfigure a given overlay graph $G^f = \left(V, E^f\right)$ to another graph $G^o = (V, E^o)$ in such a way that M (G, H) (as given in (1)) is minimized. Let $\mathcal{L}$ be the set of all of graphs that can be used as topology for the overlay network, and deg_i denotes the number of links of $peer_i$ when the graph of the overlay is G. Both the *PROP-O* algorithm reported in [6, 15], and the algorithm proposed in this paper try to find an overlay graph so that (2) is minimized subject to constraint (3).

$$\min_{G \in \mathcal{L}} \sum_{peer_i \in V} \sum_{peer_j \in V - \{peer_i\}} I(E,(i,j)) \times d_{ij} \tag{2}$$

$$s.t \quad deg_i = deg_i^f \quad \forall_{peer_i} \tag{3}$$

where deg_i^f denotes the number of links of $peer_i$ when the graph of the overlay is G^f. Constraint (3) means that the matching algorithm should not change the degree of peers. This constraint tries to hold some characteristics of the graph of the overlay network such as degree distribution and number of links. Changing the degree of peers is not appropriate in both unstructured networks and structured networks, and therefore, satisfying the constraint (3) enables the topology matching algorithm to be used in a wide range of peer-to-peer networks. To satisfy constraint (3), a local operator called the exchange operator is proposed in [6]. Figure 2 shows how the exchange operator exchanges an 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 $(peer_j, peer_k)$and$(peer_i, peer_l)$ only if $peer_i$ and $peer_j$ are adjacent to each other. With variable neighborhood radii, 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. Note that the exchange operator does not change the number of connections of the peers of the network during the exchange operation.

Existing matching algorithms such as *X-BOT* (Bias the Overlay Topology according to criteria X) [12, 17], Two-Hop-Away Neighbor Comparison and Selection (*THANCS*) [7], and Peer-exchange Routing Optimizing Protocol (*PROP*) [6, 15] suffer from two problems. 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 radii speed up the convergence of the matching algorithm (because the number of *candidate* peers at each step increases), but it increases the number of exchanges that must be endured until the convergence of the algorithm. Also, large neighborhood radii cause higher traffic and computational overhead of the network. Small neighborhood radii decreases the number of *candidate* peers at each step of the algorithm, which causes the number of exchanges to be increased. Small neighborhood radii result 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 for the operation of the local search operation may not be appropriate. The second problem is the lack of an adaptive mechanism for managing the execution of the local operator of the marching algorithm. A non-adaptive mechanism for managing the execution of the local operator leads to performing unnecessary local operations

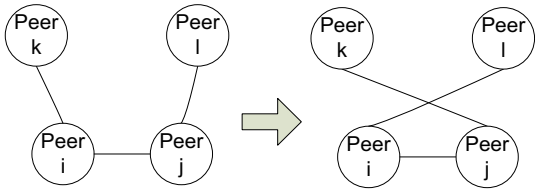

Fig. 2 Exchange Operator

(such as exchange operations) that result in increasing the overhead of the matching algorithm (higher number of peers to be reconfigured and extra control messages). It should be noted that the structure of the underlay network is fixed, but the structure of the overlay network continually changes because of the network dynamicity caused by joining and leaving peers. Therefore the topology matching algorithms try to continually observe the changes occurring in the overlay network and modify the links of the overlay network in order to mitigate the negative effects of the topology mismatching problem. Therefore, having a self-adaptive mechanism for topology matching has a positive impact on the efficiency of a matching process.

Since peer-to-peer networks are large and also dynamic, algorithms for managing them must be self-organized algorithms, and for this reason several self-organized algorithms such as *Ant Colony* (*ACO*), *Growing Neural Gas* (*GNG*), and the *Schelling Segregation Model* (*SSM*) have been reported for topology management in the literature [10, 18–20]. The *SSM*, which has been used as part of the proposed algorithm is briefly discussed in the rest of this paragraph. The *SSM* is composed of independent and identical agents [21]. 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 they have similar attributes with that agent. According to a rule called the happiness rule, each agent decides whether or not to change its neighbors. If the value of the similarity function is lower than a threshold z, the agent is unhappy and prefers to change its neighbors in order to increase the number of similar neighbors. This process continues until no agent wants to change its neighbors. In this model, designing a proper happiness rule is very crucial to the proper functioning of *SSM*.

In this paper, a self-adaptive algorithm for solving the topology mismatch problem in unstructured peer-to-peer networks will be proposed. In the proposed algorithm, we use *learning automata* to adapt the neighborhood radius and an adaptation mechanism inspired from the *SSM* to manage the execution of the exchange operator. To show the superiority of the proposed algorithm, computer experiments have been conducted and then the results are compared with the results obtained for algorithms *PROP-O* (an optimized version of *PROP* algorithm) [6, 15], *X-BOT* [12, 17], and *THANCS* [7]. The rest of this paper is organized as follows. In Sect. 2, the theory of *learning automata* is briefly described. Section 3 reviews the related works. In Sect. 4, we present a novel topology matching algorithm. Section 5 reports the results of experimentations and Sect. 6 concludes the paper.

2 Learning Automata

Learning automata (*LAs*) are models for adaptive decision making in random environments. A learning automaton has a finite set of actions. The learning process of the learning automaton is described as follows. Each time the learning automaton interacts with its environment, it randomly selects an action based on a probability vector. Initially, each action can be selected with equal probability. According to the response of the environment (reward or penalty) to the selected action, the learning automaton updates its action probability vector and then the procedure is repeated.

The updating algorithm for the action probability vector is called the learning algorithm or the reinforcement scheme. If this algorithm is properly designated, the selection probability of the appropriate action approaches unity while the probabilities of other actions approach zero. The appropriate action is an action with the highest probability of being rewarded by the environment. The interaction between the learning automaton and the random environment is shown in Fig. 3 [22].

Learning automata (*LAs*) can be classified into two main families, fixed and variable structure learning automata [22]. Variable structure learning automata, which is used in this paper, is represented by sextuple <$\underline{\beta}$, $\underline{\phi}$, $\underline{\alpha}$, P, G, T>, where $\underline{\beta}$ is a set of input actions, $\underline{\phi}$ is a set of internal states, $\underline{\alpha}$ is 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 a learning algorithm. The learning algorithm is a recurrence relation and is used to modify the probability vector.

It is evident that the crucial factor affecting the performance of the variable structure learning automata is the 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 (4) for a favorable response ($\beta = 1$) and (5) for an unfavorable response ($\beta = 0$).

$$\begin{aligned} p_i(k+1) &= p_i(k) + a(1 - p_i(k)) \\ p_j(k+1) &= p_j(k) - ap_j(k), \forall j \neq i \end{aligned} \tag{4}$$

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

The parameters a and b represent reward and penalty parameters, respectively. The parameter $a(b)$ determines the amount of increases (decreases) of the action probabilities.

3 Related Works

In the review of the relevant literature, we specifically focus on studies addressing topology mismatch problems in unstructured peer-to-peer networks. Reported algorithms can be classified into three classes of algorithms (Fig. 4): algorithms in

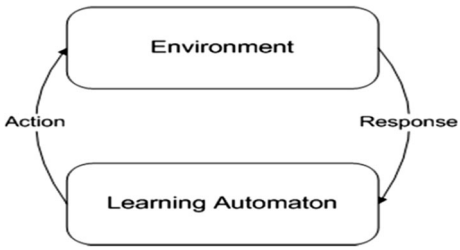

Fig. 3 Learning automaton (LA)

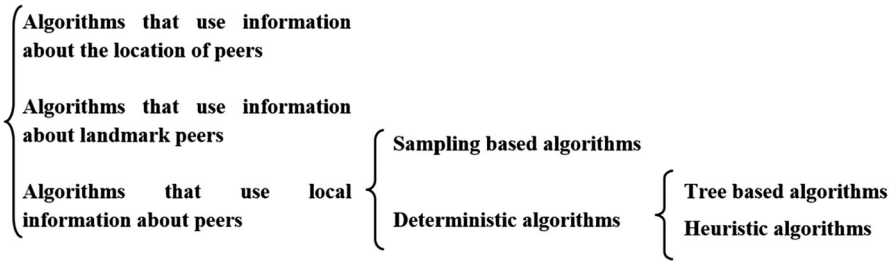

Fig. 4 A classification for topology matching algorithms

which each peer in the network uses some services such as GPS[1] or Oracle to gather information about the location of peers [23–25], algorithms in which each peer in the network uses information about landmark peers [26–33], and algorithms in which each peer in the network uses local information about its neighbors [6, 7, 9, 11, 12, 16, 17, 34–39]. In the rest of this section, we briefly review these classes of algorithms.

Algorithms that use information about the location of peers: This type of algorithm uses several services to gather some information about the locations of peers. In [23, 24], the matching algorithm utilizes GPS to gather some information about the location of peers. In [25], an algorithm is suggested that uses oracle service to gather locality information about peers. Designing algorithms based on information about the location of peers creates new challenges such as the cost and availability of required services for peers.

Algorithms that use information about landmark peers: This type of algorithm uses some information about landmark peers. Landmark peers are specific peers in the network. In these algorithms such as those reported in [26–33, 40], the peers organize themselves into clusters so that the peers that get in within a given cluster are relatively close to each other in terms of communication delay. One of the drawbacks of these algorithms is that information about landmark peers may not always be available.

Algorithms that use local information about peers: These algorithms try to optimize the overlay structures using local information about peers. These algorithms can be further classified into two categories: sampling based algorithms and deterministic algorithms, which are described in the next two paragraphs.

In sampling based algorithms, such as those reported in [9, 36, 37] some non-adaptive sampling methods are used to solve the topology mismatch problem. Since the status of peer-to-peer networks can change during the operation of the network, the non-adaptive sampling algorithms cannot be appropriate.

In deterministic algorithms, each peer uses a deterministic algorithm to find appropriate neighbors in order to solve the topology mismatch problem. Deterministic approaches can be further classified into two classes: tree based algorithms and heuristic algorithms. In tree based algorithms such as those reported in [16, 38, 39], some algorithms for finding minimum spanning trees and multicast trees are used to reconfigure the links of peers. A main drawback of tree based algorithms is that they not only need to tune their parameters with an adaptive mechanism, but they also need

[1] Global Positioning System.

another algorithm to create multicast trees or minimum spanning trees. In heuristic algorithms, some local operators such as exchange operators use some local estimation to minimize mismatched links. Since the work presented in this paper is about heuristic algorithms, in the rest of this section we specially focus on heuristic algorithms.

In heuristic algorithms, some local operators such as exchange operators are used to decrease the mismatched paths of the overlay network. This type of algorithm consists of two parts: a local search operator to gather some information about neighbors of a peer and local operators to reconfigure the connections of that peer in order to minimize its mismatched paths. *THANCS* algorithm is a heuristic algorithm reported by Liu et al. in [7]. This algorithm adds links with low delay and removes those with high delay by checking peers within its neighborhood radius. *THANCS* is a well-known topology matching algorithm. In [11], a matching algorithm is presented by *Rostami* et al. This algorithm uses local information about peers to eliminate mismatched paths. Both the algorithm reported in [11] and *THANCS* are not able to improve their functionality considering the network conditions. In [35], the Location-aware Topology Matching (*LTM*) algorithm is reported by Liu et al. In this algorithm, the peers eliminate the redundant links of the overlay by measuring the delay of the overlay links. A main drawback of this algorithm is that a synchronization method must be implemented. In [6, 15], the *PROP* was designated by Qiu et al. This algorithm has two relevant policies: Generic (*PROP-G*) and Optimized (*PROP-O*). In *PROP-G,* each peer searches its neighbors (within a particular distance) to select an appropriate peer and then exchanges all of its neighbors with the selected peer. In *PROP-O*, a threshold is defined to determine the numbers of peers that can be used in each exchange operation. In *PROP-O,* a peer searches its neighbors (within a particular distance) to select an appropriate peer and then exchanges an equal number of connections with that peer. *PROP-O* tries to preserve the number of connections of each peer during the execution of the matching algorithm. A main drawback of both *PROP-O* and *PROP-G* is that they have no self-adaptive mechanism to tune their parameters (such as parameter distance) of their local search algorithm. In [12, 17], *X-BOT* is proposed. *X-BOT* is an algorithm for topology optimization in unstructured peer-to-peer networks that can be used to solve the topology mismatch problem. This algorithm is able to bias an initial random overlay in order to optimize some efficiency criteria; for example, to reduce the delay of the overlay. This algorithm and *PROP-O* algorithm preserves the degree of nodes of the graph of the overlay network and the overlay connectivity. This algorithm has two main drawbacks. The first drawback is that this algorithm has several parameters which should be tuned by a designer. The second drawback is that the local search of this algorithm is not adaptive.

All of the heuristic algorithms reported in this section suffer from two problems. The first problem is that there is no adaptive mechanism for setting the neighborhood radius parameter that determines the scope of the local search of the matching algorithm. Finding an appropriate value for this parameter manually is a time consuming process and also error prone. Fixing this parameter is not also appropriate due to network dynamicity. The second problem is the lack of an adaptive mechanism for managing the execution of the local operator of the marching algorithm. To solve these problems a self-adaptive topology matching algorithm will be proposed in the next section.

4 The Proposed Algorithm

In this section, a self-adaptive algorithm based on the learning automata and SSM for solving the topology mismatch problem will be proposed. In the proposed algorithm, each peer searches its neighbors to find an appropriate peer, and then uses the exchange operator to exchange its connections with that peer in order to reduce the number of mismatched paths of the overlay network. We define variable r_i to save the neighborhood radius of $peer_i$. Each peer is equipped with a learning automaton. This learning automaton has two actions: "increase radius" and "decrease radius" and is responsible for adaptively tuning up the neighborhood radius. Figure 5 shows the interaction of the learning automaton LA_i of $peer_i$ with its environment when $r_i = 1$. The environment of the learning automaton LA_i residing in peer $peer_i$ is composed of the local environment of $peer_i$, and the global environment that consists of local environments of the peers, which are in the neighborhood of the $peer_i$. In the proposed algorithm, for $peer_i$, a variable called μ_i is defined and can take a value of either "not moving" or "moving". This variable determines whether $peer_i$ can participate in the exchange operation or not. Function λ_i takes information about the neighbors of $peer_i$ as the input and returns the portion of the neighbors of $peer_i$ that cannot participate in the exchange operation to all neighbors of $peer_i$. Variable μ_i and function λ_i are defined to implement the process of changing neighbors of the agents of the SSM in the proposed algorithm and are responsible to manage the execution of the exchange operator. The formal definitions of all used variables and functions are given later. Algorithm 1 shows the high level structure of the proposed algorithm. This algorithm is briefly described in the next three paragraphs.

Once $peer_i$ joins the network, it first uses a neighborhood selection algorithm such as the neighborhood selection method of *Gnutella* for finding its neighbors and then executes the proposed algorithm. Before describing the phases of the proposed algorithm, the method used in *Gnutella* for the neighborhood selection is described in more detail as follows. In this method, $peer_i$ connects to global caches, which are called the *Gnutella Web Caches*. After connecting to the caches, $peer_i$ requests a list of peers. $peer_i$ tries to connect to the peers of the list, until it connects to a specific

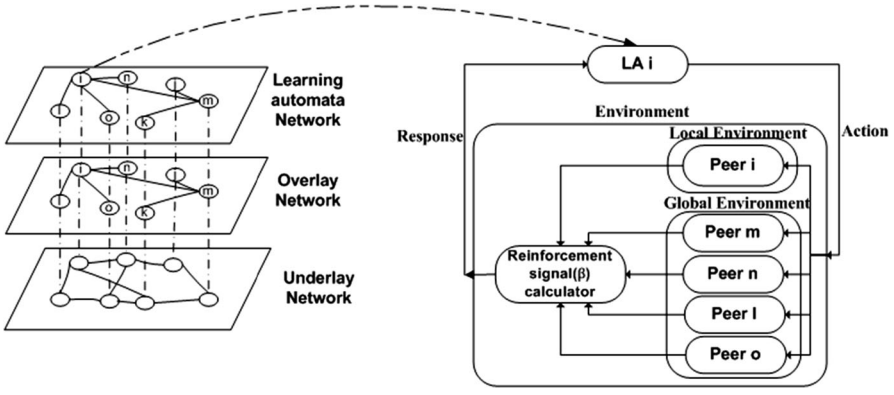

Fig. 5 Learning automaton for $peer_i$ and its environment

number of peers [1]. Now we describe the phases of the proposed algorithm that uses information about delays among peers in order to solve the topology mismatch problem. The management algorithm consists of three phases: the *local search* phase, *exchange* phase, and *maintenance* phase. The phases of the proposed algorithm are summarized in Algorithm 1 and the procedure of the reinforcement signal calculator unit used by the learning automata is given in Algorithm 2. The phases of the proposed algorithm are briefly described in the next two paragraphs.

During the *local search* phase, $peer_i$ uses its learning automaton to determine the value of the neighborhood radius. If the value of the neighborhood radius is lower than or equal to 0, the peer computes the value of the reinforcement signal (using the procedure *reinforcement_signal_calculator* given in Algorithm 2) otherwise, the peer activates all of its neighbors, exchange information about delays with its neighbors, and finds appropriate *corresponding* and *candidate* peers. Then $peer_i$ saves the found *corresponding* and *candidate* peers in its local database, computes the value of λ_i, finds the action selected by the majority of learning automata of neighboring peers, and finally computes the value of the reinforcement signal. $peer_i$ at the end of the *local search* phase asks its learning automaton to update its action probability vector and then goes to the *exchange* phase.

During the *exchange* phase, $peer_i$ first finds an appropriate *corresponding* peer, $peer_j$ from its local database to participate in the exchange operation. Then, $peer_i$ executes the exchange operator for exchanging its connections with $peer_j$, computes the value of λ_i, and sets the value of variable μ_i. The peer sets the value of μ_i to "not moving" if the value of λ_i is equal to 1 and set to "moving", otherwise. At the end of the *exchange* phase, if the value of variable μ_i is equal to "moving", $peer_i$ goes to the *local search* phase, otherwise goes to *maintenance* phase. During the *maintenance* phase, which is an ever going process, $peer_i$ continually waits for one of the events of "changing connections by a neighboring peer" or "activation of matching algorithm by neighboring peer", which in that case, $peer_i$ restarts the matching algorithm. In the next paragraph, we describe how the concept of changing neighbors of agents in the *SSM* is used in the proposed algorithm.

The process of changing neighbors in the proposed algorithm plays the same role as the process of changing neighbors of agents in the *SSM*. In the proposed algorithm, similar to the *SSM*, $peer_i$ changes its neighbors in order to increase the portion of neighboring peers (λ_i) that do not have a mismatched path with $peer_i$. Since the value of the similarity function gives valuable information about the position of $peer_i$, it is used in the stopping condition and the reinforcement signal calculator unit of the proposed algorithm. The details about the similarity function are given later in the detailed descriptions of the proposed algorithm.

Before we present the proposed algorithm in more detail we need to define the following:

1. The neighborhood radius of $peer_i$ of the algorithm is denoted by r_i. This parameter will be used to determine the broadcast scope of the control messages in the algorithm. r_i is initially set to a given value.
2. The neighborhood set of $peer_i$ denoted by $N_i^{r_i}$ contains all peers residing in the neighborhood radius of $peer_i$, that is, $N_i^{r_i} = \{peer_j \in V | dist(peer_i, peer_j) \leq r_i\}$

Algorithm 1 peer_management_skeleton()

Inputs:

m // the initial value of the neighborhood radius of the *peer*

t // the threshold for the percentage of all neighbors of the *peer* that cannot participate in the exchange operation

Notations:

Let LA determines the learning automaton of the *peer*

Let μ determines whether the *peer* can change its neighbors using exchange operation or not

Let r determines the neighborhood radius of the *peer*

Let λ determines the portion of the neighbors of the *peer* which cannot participate in the exchange operation

Let τ determines the portion of the neighbors of the *peer* which cannot participate in the exchange operation

Let A determines the selected action of LA

Let X determines the action selected by majority of learning automata of neighboring peers

Begin procedure
- $\mu \leftarrow$ "moving";
- $r \leftarrow$ m;
- $\tau \leftarrow t$;

While (μ = "moving") **Do**
  // **local search phase**
  - activate LA to choose an action and save the selected action in A;
  - change r using the action selected by LA;// the selected action of LA is saved in A
  If($r \leq 0$)**Then**
    - $\beta \leftarrow reinforcement\_signal\_calculator(r, \lambda, \tau, A, X)$;// the pseudo code of this procedure is given In Algorithm 2
  Else
    - activate all of neighbors of the *peer* determined by r;
    - exchange information about delays with neighboring peers;
    - find appropriate candidate and corresponding peers;
    - save the identifiers of found corresponding and candidate peers into a local database in the *peer*;
    - compute λ; //using information stored in the local database of the *peer*
    - find the action selected by majority of learning automata of neighbors of the *peer* and save it in X;
    - $\beta \leftarrow$ reinforcement_signal_calculator(r, λ, τ, A, X); // the pseudo code of this procedure is given in Algorithm 2
  EndIf
  - LA updates its action probability vector using reinforcement signal β;
  // **exchange phase**
  - find an appropriate corresponding peer $peer_j$ from the local database of the *peer*;
  - execute exchange operator with $peer_j$;
  - compute λ; //using information stored in the local database of the *peer* and the information of new neighbors which are added during exchange operation
  If ($\lambda = 1$)**Then**

```
26      - μ ⟵ "not moving";
27      EndIf
28    EndWhile
29    // maintenance phase
30    - wait until some event occurs;
31    If (changing connections by a neighboring peer or
          activation of matching algorithm by neighboring peers has been occurred) Then
32      - goto line 01;
33    Else
34      - goto line 29;
35    EndIf
36    Endprocedure
```

where $dist(peer_i, peer_j)$ is the length of the shortest path (with the minimum number of hops) between $peer_i$ and $peer_j$ in the overlay network. The immediate neighbors of $peer_i$ is denoted by N_i^1.

3. The 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 phase.
4. The corresponding peer set of $peer_i$ denoted by M_i. This set contains some of neighbors of $peer_i$ such as $peer_j \in N_i^{r_i}$ of which $peer_j$ has a candidate peer set $C_{ji} \subseteq N_j^1$, and $peer_i$ has a candidate peer set $C_{ij} \subseteq N_i^1$ so that

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

   In other words, M_i contains some of the neighbors of $peer_i$ that can participate in the exchange operation to decrease the number of mismatched paths of the overlay network.
5. $\lambda_i = \frac{|N_i^1| - |M_i|}{|N_i^1|}$ is the portion of neighboring peers that do not have a mismatched path with $peer_i$ (the portion of the neighbors of $peer_i$ which cannot participate in the exchange operation). The similarity function λ_i used in the proposed algorithm is designated based on the fact that when a peer has no mismatched paths with its neighbors ($|M_i| = 0$), the peer should stop executing the matching algorithm. The value of λ_i increases during the exchange operation and reaches 1 when $peer_i$ has no mismatch path ($|M_i| = 0$). The value of the similarity function gives valuable information about the position of the peer and is used for computing the reinforcement signals of the learning automata.
6. Candidate table of $peer_i$ denoted by CT_i saves information about the *corresponding peers* and the *candidate peers* of $peer_i$. This information is required by the algorithm to compute the reinforcement signal for the learning automaton of $peer_i$.
7. There are five types of control messages: *ExchangeRequest*, *ExchangeReady*, *UpdateState*, *ReleaseRequest* and *ReleaseReply* that are used in the proposed

Algorithm 2 reinforcement_signal_calculator()

Inputs:

r // the value of the neighborhood radius of the *peer*

λ // the percentage of all neighbors of the *peer* that cannot participate in the exchange operation

τ // the threshold for the percentage of all neighbors of the *peer* that cannot participate in the exchange Operation

A // the action selected by the learning automaton of the *peer*

X// the action selected by majority of learning automata of neighboring peers

Outputs:

β // the reinforcement signal of the learning automaton of the *peer*

```
Begin procedure
If(r ≤ 0)Then
  If (A = "decrease radius") Then
    - β ⟵ 0;
  Else
    - β ⟵ 1;
  End
Else
  If (λ ≥ τ) Then
    If (A = "decrease radius") Then
      - β ⟵ 1;
    Else
      - β ⟵ 0;
    EndIf
  Else
    If(X = "increase radius"and A = "increase radius")Then
      - β ⟵ 1;
    Else
      - β ⟵ 0;
    EndIf
  EndIf
EndIf
Endprocedure
```

algorithm. Table 1 shows the fields of these messages. Field Type contains the type of the message. Field TTL contains the "time to live value" for the message to ensure that the message dies out. The neighborhood radius is used to determine the TTL value. Sender and Receiver fields contain the sender or receiver identifier (such as the IP addresses). Field Action contains either the "Increase radius" or "Decrease radius". Field Candidate-Peers contains the identifiers for *candidate peers,* and field Delay contains the delays for *candidate peers*.

Now we describe the three phases of the process *LA_Match*. Algorithm 3 shows the pseudo code for process *LA_Match* performed by $peer_i$. Once $peer_i$ joins the network it uses the neighborhood selection method of *Gnutella* to find its neighbors,

Table 1 The format of the used messages

Message name	The fields of the message
ExchangeRequest	Type, Sender, TTL, Candidate-Peers
ExchangeReady	Type, Sender, Receiver, TTL, Action, Candidate-Peers, Delay
ReleaseRequest	Type, Sender, Receiver, TTL, Candidate-Peers
ReleaseReply	Type, Sender, Receiver, TTL
UpdateState	Type, Sender,TTL

broadcast *UpdateState* messages to all its neighbors, and then executes the proposed management algorithm. The detailed descriptions of the proposed algorithm are given in the remainder of this section.

4.1 Local Search Phase

The *local search* phase starts with the *initialization* phase and ends with the *candidate selection* phase as described below.

Initialization phase: During the *initialization* phase performed by $peer_i$, learning automaton LA_i randomly selects one of its actions "Increase radius" or "Decrease radius" according to its action probability vector. The new neighborhood radius r_i for $peer_i$ is then computed accordingly. If r_i is greater than or equal to one, then $peer_i$ generates an *ExchangeRequest* message that contains its immediate neighbors as *candidate peers* and broadcasts it to all of its neighbors and then goes to the *candidate selection* phase. If r_i is less than one, then $peer_i$ computes the reinforcement signal of the LA_i using the *reinforcement_signal_-calculator* procedure given in Algorithm 2. After computing the reinforcement signal, LA_i updates its probability vector and then $peer_i$ calls the *exchange* procedure to start the *exchange* phase. The pseudo code of the exchange procedure is given in Algorithm 4.

Candidate selection phase: In this phase, $peer_i$ waits for a certain duration (*SEARCH_DURATION*) to receive an *ExchangeRequest* message or *ExchangeReply* message from its neighbors. $peer_i$ upon receiving an *ExchangeRequest* message from $peer_j$, which contains the *candidate peers,* and finds the delays between itself and the *candidate peers*. $peer_i$ then sends an *ExchangeReply* message to $peer_j$. This message contains the immediate neighbors of $peer_i$, the delays between $peer_i$ and the immediate neighbors, the delays between $peer_i$ and the *candidate peers*, and the action selected by LA_i during the *Initialization phase*. $peer_i$ then restarts the *candidate selection* phase.

$peer_i$ upon receiving an *ExchangeReply* message from $peer_j$, which contains the *candidate peers* and the delays of the *candidate peers*, finds the delay between itself and the *candidate peers*. $peer_i$ then uses all of the gathered information about the delays of links to determine whether $peer_j$ can be a *corresponding peer* or not. If $peer_i$ can select some of its neighbors as C_{ij} (the candidate peer set for $peer_j$) and select some of the *candidate peers* as C_{ji} (the candidate peer set for $peer_i$) so that $|C_{ji}| = |C_{ij}|$ and $\left[\sum_{peer_k \in C_{ij}} d_{ki} + \sum_{peer_l \in C_{ji}} d_{lj}\right] > \left[\sum_{peer_l \in C_{ji}} d_{li} + \sum_{peer_k \in C_{ji}} d_{kj}\right]$ then $peer_j$ is

a *corresponding peer*. If $peer_j$ is a *corresponding peer,*then $peer_i$ inserts $peer_j$ as its *corresponding peer* and the selected peers (C_{ij} and C_{ji}) as the candidate peer sets into its candidate table. $peer_i$ then restarts the candidate selection phase. If $peer_i$ does not receive any messages during the specified period of the *SEARCH_DURATION*, then it computes the reinforcement signal for learning automaton LA_i. After computing the reinforcement signal, LA_i updates its probability vector, and then $peer_i$ goes to the *exchange* phase by calling exchange procedure.

Note that for selecting *candidate peer* sets, $peer_i$ generates all possible candidate peer sets and then compares the generated sets with each other in order to find appropriate candidate peer sets. That is, $peer_i$ selects two candidate peer sets (C_{ij} *and* C_{ji}) in such a way that the difference between the sum of the end-to-end delays of the network before and after the exchange operation becomes the maximum.

4.2 Exchange Phase

During this phase, $peer_i$ for each corresponding $peer_j$ given in its candidate table, generates a *ReleaseRequest* message containing all the *candidate peers* of $peer_j$ and then sends it to the *corresponding* $peer_j$. After sending the *ReleaseRequest* messages to all the *corresponding peers*, $peer_i$ waits for certain duration (*EXCHANGE_DURATION*) to receive the *ReleaseReply* messages or *ReleaseRequest* messages from its neighbors.

$peer_i$, upon receiving a *ReleaseRequest* message that contains *candidate peers* from $peer_j$, finds C_{ij} and C_{ji}, creates new links to the *candidate peers* that are available in C_{ji}, removes the old links to the *candidate peers* that are available in C_{ij}, sends a *ReleaseReply message to* $peer_j$, and returns to process the *LA_Match* that is called the *exchange* phase. $peer_i$ sends *ReleaseReply* messages to $peer_j$ via two paths: a path that uses the *candidate peers* that are connected to the *corresponding peer* $peer_j$ and a path that exists between $peer_i$ and $peer_j$ to ensure that *ReleaseReply* messages reach $peer_j$.

$peer_i$, upon receiving a *RelaseReply* from $peer_j$, goes to its candidate table and finds those *candidate peers* whose *corresponding peer* is $peer_j$. Then it finds C_{ij} and C_{ji}, creates new links to the *candidate peers* that are available in C_{ji}, removes the old links to the *candidate peers* that are available in C_{ij}, sends an *UpdatedState* message to its neighbors, and returns to the process *LA_Match* that is called the *exchange* phase. If $peer_i$ does not receive any messages during the specified period of *EXCHANGE_DURATION* then it returns to process *LA_Match* that is called the *exchange* phase.

When $peer_i$ returns to the *LA_Match* process, if the portion of its neighbor that cannot participate in the exchange operation is equal to one then its variable μ_i is set to "not moving" and it goes to the *maintenance* phase. $peer_i$ restarts the *local search* phase when its variable μ_i is equal to "moving".

4.3 Maintenance Phase

During this phase, $peer_i$ waits until some event occurs. $peer_i$ restarts the matching algorithm if it receives *UpdateState* or *ExchangeRequest* messages and restarts the *maintenance* phase, otherwise.

Algorithm 3 LA_Match

Inputs:

m// the initial value of the neighborhood radius

t // the threshold for the percentage of all neighbors of the *peer* that cannot participate in the exchange operation

Notations:

Let $r$determines the neighborhood radius of the *peer*

Let μ determines whether the *peer* can participate in exchange operation or not

Let λ determines the portion of the neighbors of the *peer* which cannot participatein the exchange operation

Let CT determines the candidate table of the *peer*

Let LA determines the learning automaton of the *peer*

Begin procedure
- $\mu \longleftarrow$ "moving";
- $r \longleftarrow$ m;
- $\tau \longleftarrow t$;

While ($\mu =$ "moving") **Do**
//**local search phase**
//**Initialization Phase**
- activate LA to select an action and save it in A;

If($A =$ "$increase\ radious$")**Then**
- $r \longleftarrow r + 1$;

Else
- $r \longleftarrow r - 1$;

EndIf
If($r \leq 0$)**Then**
- $\beta \longleftarrow$ reinforcement_signal_calculator(r, λ, τ, A, X); // the pseudo code of this procedure is given In Algorithm 2

Else
- Broadcast *ExchangeRequest* message containing the immediate neighbors to all the neighbors determined by r;

// **Candidate selection phase**
- isSelectionCompleted $\longleftarrow$ False;

Repeat
- wait for a certain duration *SEARCH_DURATION* for *ExchangeRequest* or *ExchangeReply* message;

If(*no message* has been received during *SEARCH_DURATION*) **Then**
- isSelectionCompleted $\longleftarrow$ True;

Else
If(*ExchangeRequest* message has been received from another $peer_j$) **Then**
- find delays to ***candidate peers***;//***candidate peers*** are reported by *ExchangeRequest* messages

- select all immediate neighbors and consider them as ***candidate peers***;
- send an *ExchangeReply* message containing the selected ***candidate peers***, delays and action A to $peer_j$;// A is the action chosen by LA during Initialization phase

EndIf

If (an *ExchangeReply* message has been received from $peer_j$)**Then**

- find delays to ***candidate peers***;//***candidate peers*** are reported by *ExchangeReply* message
- select candidate peer sets;//select candidate peer sets using all delays which are found and reported by *ExchangeReply* message

If($the\ candidate\ peer\ sets\ are\ not\ empty$)**Then**

- insert $peer_j$ as the ***corresponding peer*** and the selected candidate peer sets of $peer_j$ in candidate table CT;

EndIf

EndIf

EndIf

Until (isSelectionCompleted**)**

- compute λ ; //using CT
- find the action selected by majority of learning automata of neighboring peers and save it in X; // X is found using information provided by *ExchangeReply* messages
- $\beta \longleftarrow \text{reinforcement\_signal\_calculator}(r, \lambda, \tau, A, X)$; // the pseudo code of this procedure is given in Algorithm 2

EndIf

- LA updates its action probability vector using reinforcement signal β;

// Exchange phase

- call exchange$(CT, peer)$; // the pseudo code of this procedure is given in Algorithm 4
- sends an *UpdatedState* message to the neighbors of the *peer*;

If ($\lambda = 1$)**Then**

- $\mu \longleftarrow$ "not moving";

EndIf

EndWhile

// maintenance phase

- wait until some events occurs;

If (an *UpdateState* or *ExchangeRequest* message has been received)**Then**

- goto line 01;

Else

- goto line 51;

EndIf

Endprocedure

Algorithm 4 Exchange

Inputs: $CT_i, peer_i$

Begin procedure
If (CT_i is not empty)**Then**
  For each ***corresponding peer*** $peer_j$ in the CT_i **do**
    - generate a *ReleaseRequest* message containing all the ***candidate peers*** of $peer_j$ and send it to $peer_j$;
  EndFor
  - isExchangeCompleted ⟵ False;
  Repeat
    - Wait for a certain duration *EXCHANGE_DURATION* for *ReleaseReply* or *RelaseRequest* message;
    If(*no message* has been received during *EXCHANGE_DURATION*)**Then**
      - isExchangeCompleted ⟵ True
    Else
      If(a *RelaseRequest* message has been received from another $peer_j$) **Then**
        - find C_{ij} and C_{ji};
        - creates new links to the ***candidate peers*** in C_{ji};
        - send a *ReleaseReply* message to $peer_j$;
        - removes the old links to the ***candidate peers*** in C_{ij};
      EndIf
      If(*RelaseReply* message has been received from a $peer_j$) **Then**
        - select all ***candidate peers*** whose ***corresponding peer***s is $peer_j$;
        - find C_{ij} and C_{ji} ;
        - creates new links to the ***candidate peers*** in C_{ji};
        - removes the old links to the ***candidate peers*** in C_{ij};
      EndIf
  EndIf
  Until (isExchangeCompleted)
EndIf
Endprocedure

5 Experimental Results

All simulations have been implemented using *OverSim* that is an overlay network simulation framework [41, 42]. *OverSim* supports different kinds of underlay network models: *Simple*, *SingleHost*, *INET,* and *ReaSE* [43]. *Simple model* is the most scalable one and the *ReaSE model* is able to generate different types of underlay networks. We used the *ReaSE model* to generate router-level topologies [44]. Experiments reported in this section are conducted on three different underlying networks: *Topology.1* as an example of the *Simple model*, and *Topology.2* and *Topology.3* as examples of the *ReaSE model,* as given in Table 2. For the overlay network management algorithm, *Gnutella* [45] and *GIA* [2] are used. Since *Gnutella* is used to test several matching algorithms reported in [6, 7, 9, 36–39, 46] and it is a peer-to-peer network with an open and well-documented protocol

specification, we use it as the default overlay network management algorithm in this section. We used the *Gnutella* dataset of [47] to generate the initial overlay topology. This data set contains a graph where its nodes, edges, and diameter are 10,876, 39,994 and 9, respectively. In an experiment, *GIA* is used as the overlay network management algorithm. *GIA* is fully implemented in *Oversim*. The Random churn model [41] and Pareto churn model [41] are also used to model the process of joining and leaving peers in the network.

To evaluate the performance of the proposed algorithm called *LA_Match,* it is compared with three algorithms: *PROP-O* [6, 15], *X-BOT* [12, 17], and *THANCS* [7]. It should be noted that, the proposed algorithm, like algorithms *PROP-O*, *X-BOT*, and *THANCS* (to which the proposed algorithm is compared) uses only the delays of links for solving the topology matching problem. Other algorithms, such as those reported in [23–33, 40] use other information, such as geographical positions and information about landmark peers, which may not always available. Both *PROP-O* and *X-BOT* algorithms, like the proposed algorithm use a management operator (called the exchange operator) to exchange connections among the peers to maintain the number of connections of each peer during the execution of the matching algorithm. Other algorithms, such as those reported in [9, 11, 16, 35–39] change the number of connections of peers that are not appropriate for some peer-to peer networks. The *PROP-O* algorithm has a parameter called *nhops* that plays the same role as parameter *m* in the proposed algorithm. The reason of selecting *THANCS* is that it is a well-known algorithm for topology matching in unstructured peer-to-peer networks.

The proposed algorithm has four parameters: *m*, *t*, *SEARCH_DURATION*, *and EXCHANGE_DURATION. T*wo parameters *SEARCH_DURATION* and *EXCHANGE_DURATION* must be set to large values in order to provide enough time for the peers of the network to communicate with their neighbors. For the experiments, *SEARCH_DURATION* is set to 5*s,* and *EXCHANGE_DURATION* is set to 5*s*. To test the impact of parameters *m* and *t* on the performance of the algorithm and also to show the learning capability of the proposed algorithm, several experiments were conducted as reported in the remaining part of this section. For all experiments, each peer is equipped with a variable structure learning automaton of type L_{RP} with a

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 [48, 49] data
Topology.2	Consists of 10 autonomous systems, and about 1087 router—level nodes. This topology contains few and populated group
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

reward parameter $a = 0.25$ and a penalty parameter $b = 0.25$. The other parameters of the *PROP-O* algorithm are set as follows: *MIN_VAR* = 0, *MAX_INIT_-TRIAL* = 10, $m = 3$, *INIT_TIMER* = 10 min, *MAX_TIMER* = 1000 min. The parameters of the *X-BOT* algorithm are set as follows: $k = 6$, *PBO* = 10 s, $\pi = 2$, *and* $\mu = 3$. In the *THANCS* algorithm each peer flood control messages periodically (every 5 s) to gather required information about its neighbors. Experiments 1, 2, and 3 are conducted to find the most appropriate parameters for the proposed algorithm to be used in the later experiments. Results reported are averages over ten different runs.

The algorithms are compared with respect to four metrics: overlay communication delay, mismatched paths, overlay reconfiguration overhead, and control message overhead. 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).
- *Number of Mismatched Paths* (*MP*) is the number of all mismatched paths of the overlay network. A mismatched path is a path in an overlay network that in its corresponding underlay path, a particular position has been visited more than once.
- *Overlay reconfiguration overhead* (*ORO*) is the number of peers that are reconfigured by a matching algorithm. This metric implicitly shows the changes made by the local operators (such as exchange operators) of the matching algorithms.
- *Control message overhead* (*CMO*) is the number of extra control messages generated for the purpose of reconfiguration of an overlay network.

5.1 Experiment 1

This experiment is conducted to study the effect of parameter t on the performance of the proposed algorithm when parameter m is set to 1. For this study, the proposed algorithm 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 the four above mentioned criteria. According to the results of this experiment, which are shown in Fig. 6, one may conclude the following:

- In terms of *OCD* and *MP*, the proposed algorithm (with $t = 0.6$ or $t = 0.9$) performs better than the proposed algorithm with $t = 0.3$. This is because a low value for parameter t leads to low sensitivity of the proposed algorithm of a peer to the information about other peers of the network. It should be noted that in each peer, parameter t implicitly determines whether or not the information about neighboring peers should be considered in the reinforcement signal calculator unit of the proposed algorithm. Since the knowledge about the neighbors of the peers gives valuable information about the state of the network, a low value for parameter t lead to conduct the learning automata of the peers to converge to inappropriate actions which lead to a low accuracy of the proposed

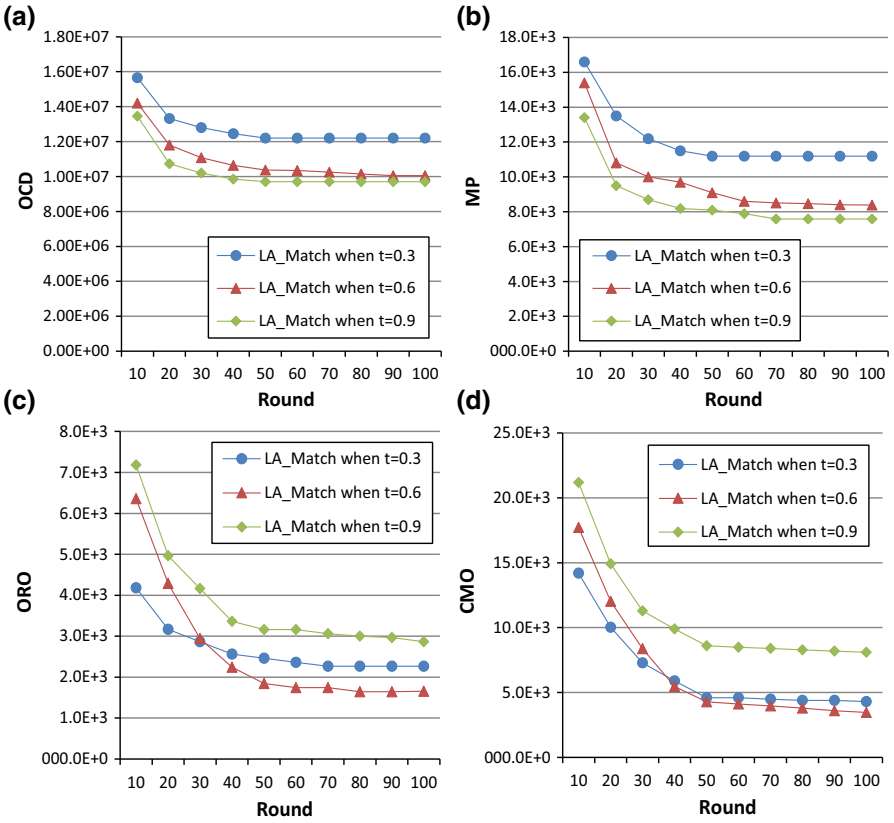

Fig. 6 The effect of parameter t on the performance of *LA_Match* with respect to **a** *OCD*, **b** *MP*, **c** *ORO*, **d** *CMO*

algorithm (with respect to a high *OCD* and *MP*) and high overhead caused by the algorithm (with respect to a high *CMO* and *ORO*).

- In terms of *ORO* and *CMO,* the proposed algorithm with $t = 0.6$ performs better than the proposed algorithm with $t = 0.3$ or $t = 0.9$ except for the early rounds of the simulation.
- Increasing the value of parameter t leads to a decrease in *OCD* and *MP*, but increases *ORO* and *CMO* in the early rounds of the simulation. This means that by increasing the value of parameter *t,* we can find an appropriate overlay with a low *OCD* (overlay communication delay) and a low *MP* (number of mismatched paths). However, we will have a high overhead (with respect to *CMO* and *ORO*) in the early rounds of the simulation.
- From round 40 to round 100, the proposed algorithm (when $t = 0.9$) does not perform well with respect to *CMO* and *ORO* but does perform well with respect to *OCD* and *MP*. In other words, for a high value for parameter t, when the overlay topology is changed to an appropriate overlay (with respect to a low *OCD* and *MP*), the overhead of the proposed algorithm remains high (with

respect to a high *CMO* and *ORO*). This is because the learning automata of the peers do not decrease the parameters neighborhood radius of peers even after changing the overlay topology to an appropriate one. A high value for parameter t leads to high sensitivity of the proposed algorithm to information about peers of the network. Since the knowledge about the neighbors of some peers of the network may not reflect the whole state of the network, a high value for parameter t may conduct the learning automata of some peers of the network to converge to inappropriate actions that lead to set the neighborhood radius of the peers to large values. It is obvious that large values for the parameter neighborhood radius lead to generating many control messages and performing many unnecessary changes in the network that leads to a high *CMO* and *ORO*.

5.2 Experiment 2

This experiment is conducted to show the impact of the parameter m (neighborhood radius) on the performance of the proposed algorithm when the parameter t is set to 0.6. For this purpose, the proposed algorithm is tested for three initial values for parameter $m = 1$, 2, and 3 and *Topology.1* is used as underlay topology. The results are compared with respect to the four above mentioned criteria. According to the results of this experiment that are shown in Fig. 7, one may conclude the following:

- Increasing the value of parameter m leads to improving the performance of the proposed algorithm in terms of *OCD* and *MP*. This is because a large radius gives each peer of the overlay more appropriate *corresponding* and *candidate* peers during a local search. Appropriate *corresponding* and *candidate* peers can participate in an exchange operation to decrease more mismatched paths that lead to low *OCD* and *MP*.
- Increasing the value of parameter m results in high *CMO* and *ORO* at early rounds of the simulation. Note that the parameter m affects the scope of the local search in the proposed algorithm. In a peer, parameter m determines a set of peers so that the information about them should be used by the reinforcement calculator unit of the matching algorithm in that peer. Increasing the value of parameter m results in gathering more information about the current state of the network in an early round of the simulation. Gathering more information results in generating more control messages (with respect to a high *CMO*) and more peers can be reconfigured (with respect to a high *ORO*). Note that, gathering more information about the network enables each peer to find appropriate links to other peers in order to solve the topology mismatch problem which leads to a low *OCD* and *MP*.
- It can be noted from the results that as the time passes the performance of the proposed algorithmin in terms of *CMO* and *ORO* improves. This is because, after finding appropriate neighbors by a peer, the peer is able to decrease its neighborhood radius using its own learning automata. Decreasing the neighborhood radius leads to a decrease in the scope of the local search, and decreasing the scope of the local search leads to a low *CMO* and *ORO*.

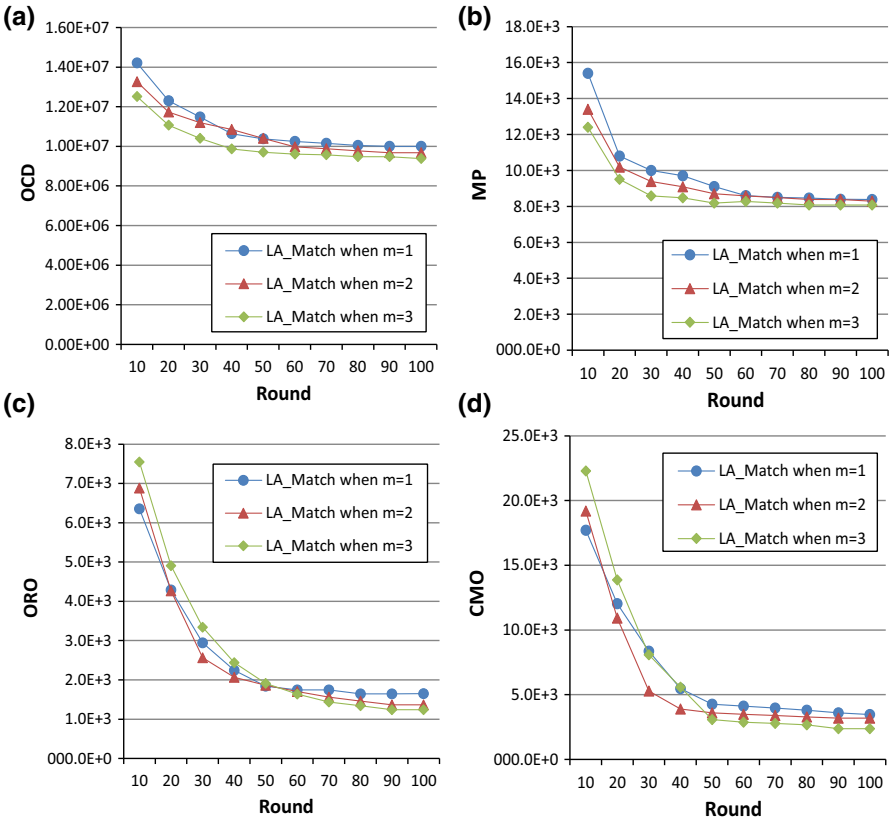

Fig. 7 The effect of parameter *m* on the performance of *LA_Match* with respect to **a** *OCD*, **b** *MP*, **c** *ORO*, **d** *CMO*

5.3 Experiment 3

This experiment is conducted to compare the proposed algorithm with (1) the proposed algorithm in which the *learning automaton* residing in each peer is deactivated (*LA_Match_D algorithm*); this is the proposed algorithm without leaning capability; (2) the proposed algorithm in which each *learning automaton* is replaced with a *learning automaton* that has three actions, "increase radius", "don't change radius", and "decrease radius" (*LA_Match_M algorithm*); and (3) the proposed algorithm in which each *learning automaton* is replaced with a *pure chance automaton*(*LA_Match_P algorithm*). In a *pure chance automaton* the actions of automaton are always selected with equal probabilities [50]. For this experiment, parameter *m* is set to 1 and parameter *t* is set to 0.6 and *Topology.1* is used as the underlay topology. The results of this experiment are given in Fig. 8. From the result we may conclude the following:

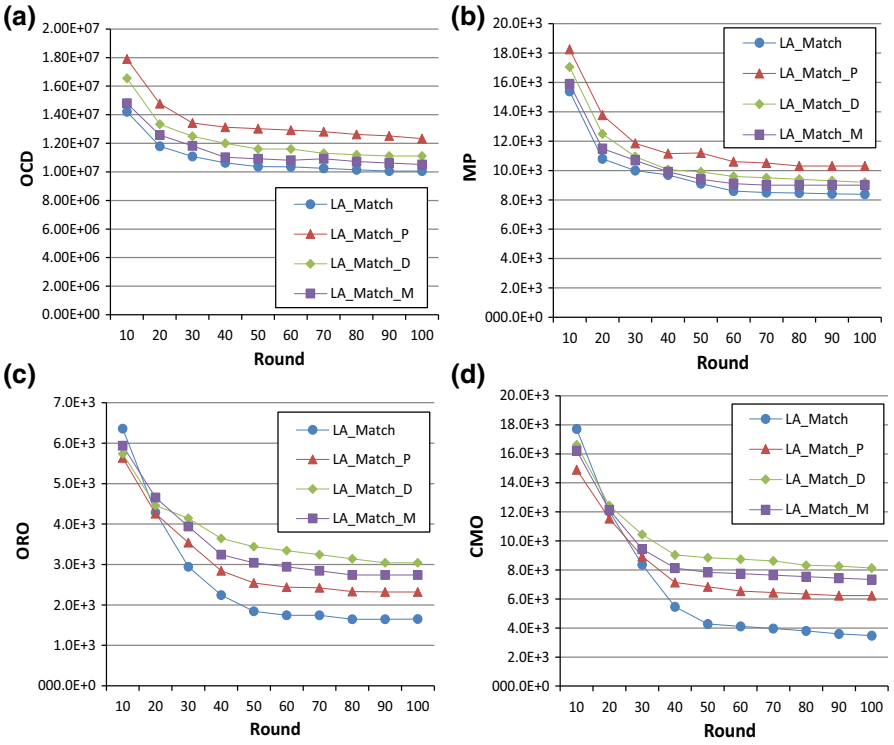

Fig. 8 Comparison of different versions of the proposed algorithm with respect to **a** *OCD*, **b** *MP*, **c** *ORO*, **d** *CMO*

- In terms of *OCD*, *MP*, *CMO,* and *ORO*, the *LA_Match* and *LA_Match_M* algorithms perform better than other algorithms (*LA_Match_D and LA_Match_P*) except for the early periods of the operation of the network. This is because of the fact that in *LA_Match* and *LA_Match_M* algorithms (because of their learning capability) improve their performance by updating their action probabilities according to the reinforcement signal received from the network. This reinforcement signal, which is computed based on some information about the peers and their neighbors reflects the current state of the network, and it is used to improve the behavior of the *learning automata* in finding the appropriate neighborhood radius. Finding the appropriate neighborhood radius leads to finding better *candidate* peers and *corresponding* peers resulting in fewer numbers of mismatched paths and also a lower *OCD, CMO* and *ORO*.
- *LA_Match* performs better than *LA_Match_M* in terms of *CMO* and *ORO*. This is because in *LA_Match_M*, the *learning automaton* of a peer may converge to "don't change radius", which leads to fixing the neighborhood radius for the peer. When the neighborhood radius of a peer is fixed and the peer cannot find an appropriate position within its neighborhood, then many unnecessary control

messages will be generated by the peer (leading to a higher *CMO*) or unnecessary changes may occur in connections of the peer with other peers (leading to an increase *ORO*).

- From the results of this experiment we may conclude that the learning automata with two actions ("increase radius" and "decrease radius") can be sufficient for this problem. This is because the stopping condition used in the *LA_Match* can conduct the process of matching without using additional action for the learning automata. In the *LA_Match*, the matching algorithm tries to change its neighborhood radius when it has a mismatched path with its neighbors. In a $peer_i$, the value of λ_i is equal to a portion of the neighboring peers that haven't the mismatched path to the $peer_i$, and it is used in the stopping condition of the algorithm. When $peer_i$ has no mismatched path with its neighbors, the matching algorithm is stopped and $peer_i$ does not need to execute the matching algorithm so it isn't forced to learn a specific neighborhood radius. Therefore, a learning automaton with two actions ("increase radius" and "decrease radius") can be sufficient for this problem.

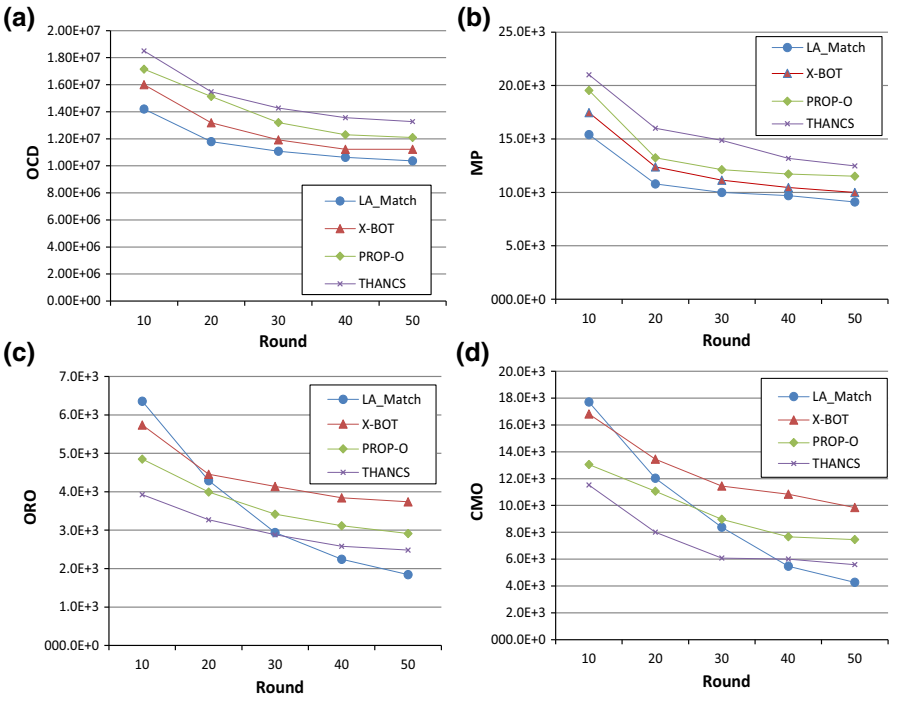

Fig. 9 Comparison of *LA_Match* with existing matching algorithms with respect to a **a** *OCD*, **b** *MP*, **c** *ORO*, **d** *CMO*

5.4 Experiment 4

In this experiment, we compare the proposed algorithm with *X-BOT*, *PROP-O,* and *THANCS* algorithms with respect to *OCD*, *MP*, *CMO,* and *ORO*. In this experiment, parameters *m* and *t* of the proposed algorithm are set to 1 and 0.6 respectively, and *Topology.1* is used as the underlay topology. According to the results of this experiment which are shown in Fig. 9, we may conclude that in term of *OCD*, *MP*, *CMO,* and *ORO*, the proposed algorithm performs better than *X-BOT*, *PROP-O,* and *THANCS* algorithms except for the early periods of the operation of the network. This is because of the fact that in the proposed algorithm, each peer utilizes a *learning automaton* for parameter adaptation (parameter neighborhood radius) and a mechanism inspired from the *SSM* for adaptive execution of the exchange operator that improves its functionality as the network operation proceeds. Adaptation of parameter *m* helps each peer to find appropriate *corresponding* and *candidate* peers to be used in the exchange operation leading to a lower *OCD* and *MP*. Adaptive execution of the exchange operator also helps each peer to manage the exchange operators in such a way that unnecessary exchange operations are avoided and lead to a lower *ORO* and *CMO*. In contrast to *X-BOT, PROP-O,* and *THANCS* algorithms, which do not utilize adaptive mechanisms to improve their functionality, the proposed algorithm (by utilizing an adaptive mechanism) is able to

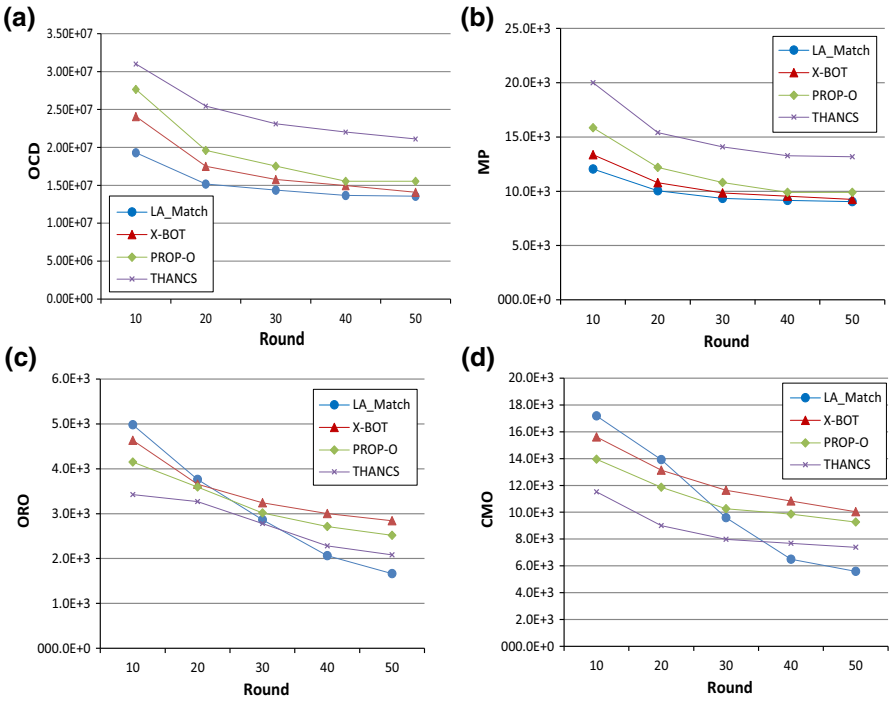

Fig. 10 Comparison of *LA_Match* with existing matching algorithms with respect to **a** *OCD*, **b** *MP*, **c** *ORO*, **d** *CMO* when *GIA* is used

improve the quality of the overlay (with respect to a low *OCD* and *MP*) and decrease its overhead (with respect to a low *ORO* and *CMO*).

5.5 Experiment 5

In this experiment, we study the performance of the proposed algorithm when $m = 1$ and $t = 0.6$ on an overlay that utilizes *GIA* to manage the overlay topology. In this experiment, *Topology.1* is used in the underlay network. The results are compared with the results obtained for *X-BOT*, *PROP-O,* and *THANCS* algorithms in terms of *OCD*, *MP*, *CMO,* and *ORO*. In this experiment, each peer uses *GIA* to find its neighbors and then executes the matching algorithm (*LA_Match*, *PROP-O*, *X-BOT*, *and THANCS*) to modify its connections to its neighbors in order to solve the topology mismatch problem. From the results given in Fig. 10, we can say that the proposed algorithm performs better than other algorithms in terms of *OCD*, *MP*, *CMO*, and *ORO* except for the early rounds of the simulation. This means that, the proposed algorithm can be efficient even for *GIA*. This is because the proposed algorithm is equipped with a self-adaptive mechanism and its design is not dependent on a specific overlay network, and therefore, it can be useful for different types of overlay networks.

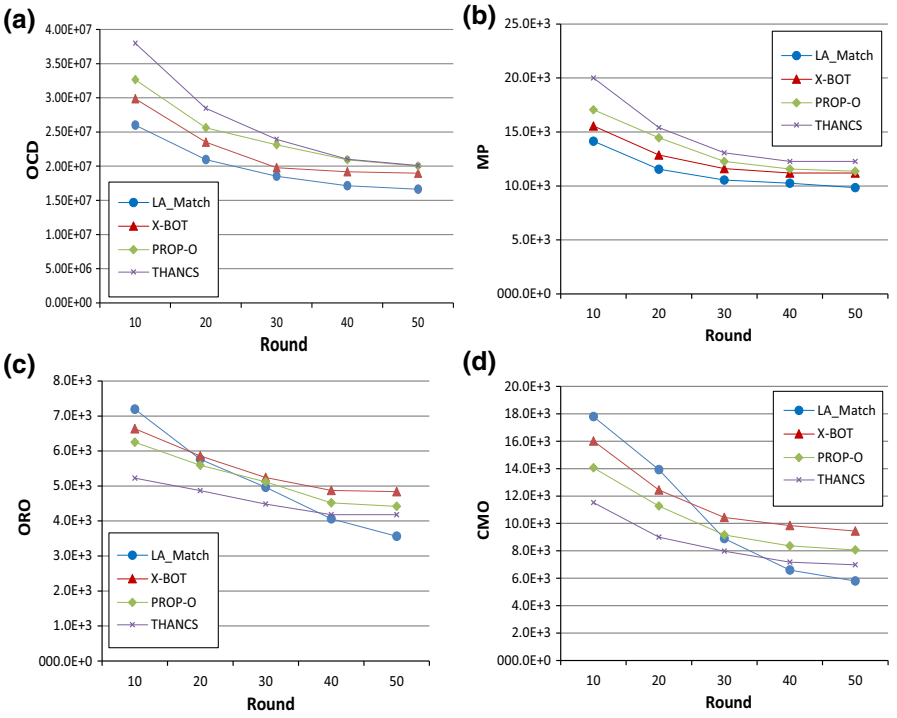

Fig. 11 Comparison of *LA_Match* with existing matching algorithms with respect to **a** *OCD*, **b** *MP*, **c** *ORO*, **d** *CMO* when *Topology.2* is used

5.6 Experiment 6

In this experiment, we compare the proposed algorithm with *PROP-O*, *X-BOT*, *and THANCS* algorithms when topologies *Topology.2* and *Topology.3* have been used. Both *Topology.2* and *Topology.3* belong to the class of router level topologies [44]. *Topology.2* contains few 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 words, in *Topology.3* the probability that two peers in the overlay network belong to different autonomous systems is high. Parameters m and t of the proposed algorithm are set to 1 and 0.6, respectively. The results are compared with respect to four criteria *OCD*, *MP*, *CMO,* and *ORO*. Figure 11 give the results of this experiment for *Topology.2* and Fig. 12 for *Topology.3.* From the results one may conclude the following:

- In terms of *OCD* and *MP*, the proposed algorithm outperforms *X-BOT*, *PROP-O*, *and THANCS* algorithms when *Topology.3* is used. This is because (1) for *Topology.3* the probability that any pair of peers in the overlay network selected by the exchange operator be far apart from each other is high and (2) the proposed algorithm is able to find more appropriate *candidate* peers due to the

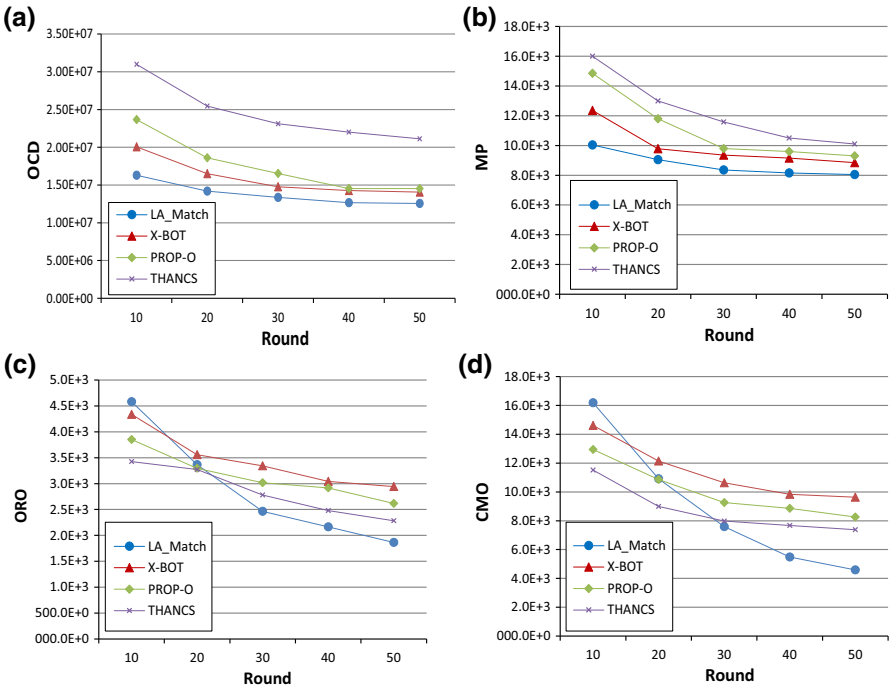

Fig. 12 Comparison of *LA_Match* with existing matching algorithms with respect to **a** *OCD*, **b** *MP*, **c** *ORO*, **d** *CMO* when *Topology.3* is used

adaptation of the parameter radius, and therefore, the exchange operator of the proposed algorithm can significantly reduce both *OCD* and *MP*.

- In terms of *OCD* and *MP*, the proposed algorithm performs better than *X-BOT*, *PROP-O,* and *THANCS* algorithms when *Topology.2* is used. Since *Topology.2* has large and populated groups, for other matching algorithms that use local search with fixed neighborhood radii take some number of rounds until a peer finds an appropriate candidate to be exchanged with. This leads to a higher *OCD* and *MP*.

Table 3 Churn model setting

Churn model type	Churn model name	Churn model parameter
Random churn model [41]	Random churn 1	joining_probability = 0.8
		leaving _probability = 0.2
Random churn model [41]	Random churn 2	joining _probability = 0.7
		leaving _probability = 0.3
Pareto Churn model [41]	Pareto Churn 1	Lifetime mean: 90 s
		Deadtime mean: 10 s

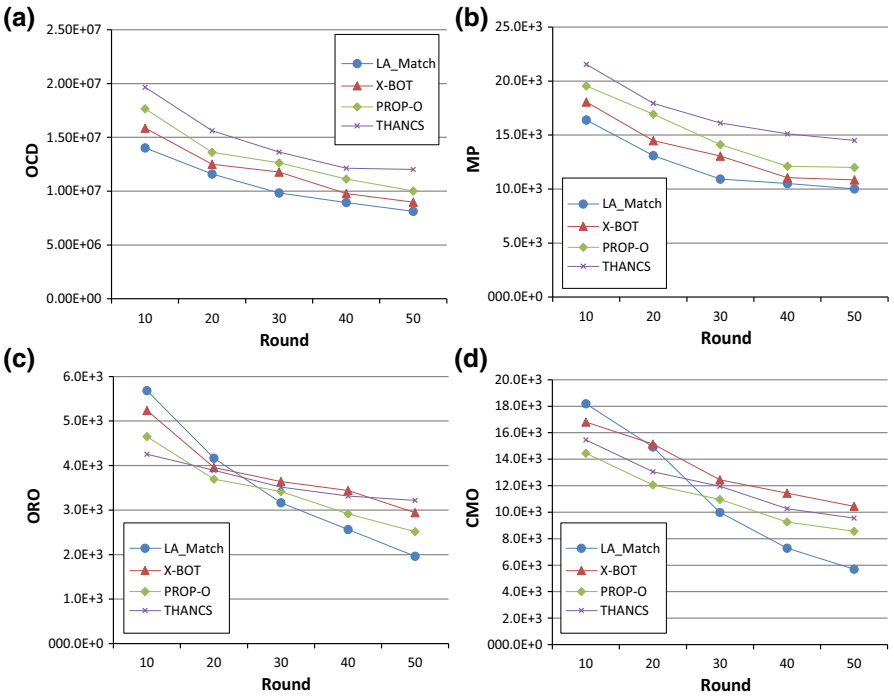

Fig. 13 Comparison of *LA_Match* with existing matching algorithms with respect to **a** *OCD*, **b** *MP*, **c** *ORO*, **d** *CMO* when *Random churn 1* is used

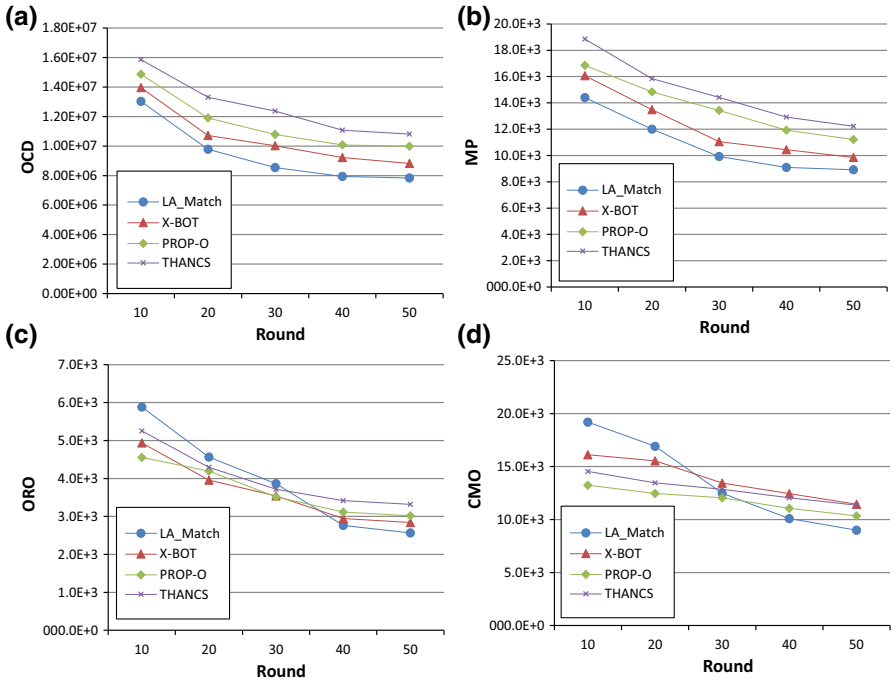

Fig. 14 Comparison of *LA_Match* with existing matching algorithms with respect to **a** *OCD*, **b** *MP*, **c** *ORO*, **d** *CMO* when *Random churn 2* is used

- From the results of this experiment, we can conclude that the proposed algorithm can be efficient even for router level topologies. This is because the proposed algorithm uses some information about delays of links, and delays of the link reflect information about the structure of the underlay network. Therefore, the overlay network utilizing the proposed algorithm is able to change its configuration considering information about the underlay network.

5.7 Experiment 7

This experiment is conducted to study the effect of different churn models on the performance of the proposed algorithm when $m = 1$ and $t = 0.6$. The churn models used are the Random churn model [41] and Pareto churn model [41] according to Table 3. In Table 3, the probabilities of joining and leaving the peers for two Random churn models and the lifetime mean and the dead time mean for one Pareto churn model are given. The results obtained for the proposed algorithm are compared with the results for the *X-BOT*, *PROP-O,* and *THANCS* algorithms with respect to *OCD*, *MP*, *ORO*, and *CMO*. Figures 13, 14 and 15 give the results of this experiment.

From the result we may conclude the following:

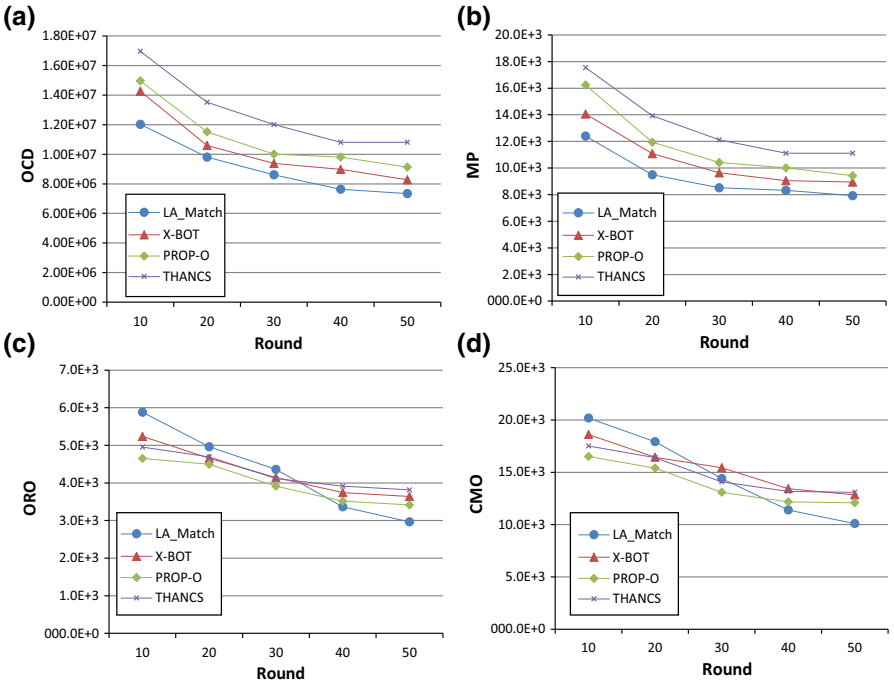

Fig. 15 Comparison of *LA_Match* with existing matching algorithms with respect to **a** *OCD*, **b** *MP*, **c** *ORO*, **d** *CMO* when *Pareto Churn 1* is used

- In terms of overlay communication delay (*OCD*) and number of mismatched paths (*MP*) the proposed algorithm performs better than *PROP-O, X-BOT,* and *THANCS* algorithms for all churn models. This is because of the fact that the proposed algorithm tunes up its neighborhood radius in a self-adaptive manner that leads to finding better *candidate* peers and therefore, results in a fewer number of mismathed paths and also a lower *OCD*.
- In terms of control message overhead (*CMO*) and overlay reconfiguration overhead (*ORO*), the proposed algorithm performs worse than other algorithms at early rounds of the simulation for all the churn models. This is because in the proposed algorithm, each peer cooperates with its neighbors to improve the decisions of its management unit during the operation of the network. The information about the joining and leaving peers will be propagated among peers because of the cooperation among peers. The information propagation among peers increases the amount of control messages of the algorithm (with respect to a high *CMO*) and motivates the management units of more peers to consider the changes that were made in the network and which lead to an increase in the changes of the configuration of the peers (with respect to a high *ORO*). It should be noted that *X-BOT*, *PROP-O*, and *THANC* have no mechanism to mitigate the negative effects of the churn.

6 Conclusion

In this paper, an algorithm based on *learning automata* and the *SSM* to solve the topology mismatch problem was proposed. The proposed algorithm uses *learning automata* to improve its local search operation and the *SSM* to improve its connection establishment process. Improving the local search operation causes more appropriate *corresponding* and *candidate* peers to be found and used by the exchange operation leading to decreasing the number of mismatched paths and also decreasing the delays of the paths of the overlay network. Improving the connection establishment process helps each peer to better manage the exchange operators that prevent performing unnecessary exchange operations that lead to decreasing the number of reconfigured peers and the number of control messages of the matching algorithm. To evaluate the proposed algorithm, several computer experiments have been conducted using *OverSim*. Experimental results showed that the proposed algorithm can compete with some of the existing matching algorithms including *X-BOT*, *PROP-O,* and *THANCS* algorithms in terms of overlay communication delay and the number of mismatched paths.

References

1. Kwok, Y.K.: Peer-to-Peer Computing: Applications, Architecture, Protocols, and Challenges. CRC Press, Boca Raton (2011)
2. Chawathe, Y., Ratnasamy, S., Breslau, L., Lanham, N., Shenker, S.: Making gnutella-like p2p systems scalable. In: Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, pp. 407–418. ACM, Karlsruhe (2003)
3. Clarke, I., Sandberg, O., Wiley, B., Hong, T.: Freenet: a distributed anonymous information storage and retrieval system. In: Designing Privacy Enhancing Technologies, pp. 46–66. Springer, Berkeley (2001)
4. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: a scalable peer-to-peer lookup service for internet applications. ACM SIGCOMM Comput. Commun. Rev **31**, 149–160 (2001)
5. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content-addressable network. ACM SIGCOMM Comput. Commun. Rev **31**, 161–172 (2001)
6. Qiu, T., Chan, E., Ye, M., Chen, G., Zhao, B.Y.: Peer-exchange schemes to handle mismatch in peer-to-peer systems. J. Supercomput **48**, 15–42 (2009)
7. Liu, Y.: A two-hop solution to solving topology mismatch. IEEE Trans. Parallel Distrib. Syst. **19**, 1591–1600 (2008)
8. Liu, Y., Zhuang, Z., Xiao, L., Ni, L.M.: A distributed approach to solving overlay mismatching problem. In: Proceedings of the 24th International Conference on Distributed Computing Systems, pp. 132–139. IEEE Computer Society, Tokyo (2004)
9. Hsiao, H.C., Liao, H., Yeh, P.S.: A near-optimal algorithm attacking the topology mismatch problem in unstructured peer-to-peer networks. IEEE Trans. Parallel Distrib. Syst. **21**, 983–997 (2010)
10. Singh, A., Haahr, M.: Decentralized clustering in pure p2p overlay networks using Schelling's model. In: IEEE International Conference on Communications, pp. 1860–1866. IEEE Computer Society, Glasgow (2007)
11. Rostami, H., Habibi, J.: Topology awareness of overlay P2P networks. Concurr. Comput. Pract. Exp **19**, 999–1021 (2007)
12. Leitão, J., Marques, J.P., Pereira, J., Rodrigues, L.: X-bot: a protocol for resilient optimization of unstructured overlay networks. IEEE Trans. Parallel Distrib. Syst. **23**, 2175–2188 (2012)

13. Ren, S., Guo, L., Jiang, S., Zhang, X.: SAT-Match: a self-adaptive topology matching method to achieve low lookup latency in structured p2p overlay networks. In: Parallel and Distributed Processing Symposium, pp. 83–94. IEEE Computer Society, Santa Fe (2004)
14. Qiu, T., Wu, F., Chen, G.: A generic approach to make structured peer-to-peer systems topology-aware. In: Third International Symposium on Parallel and Distributed Processing and Applications, pp. 816–826. Springer, Nanjing (2005)
15. Qiu, T., Chen, G., Ye, M., Chan, E., Zhao, B.Y.: Towards location-aware topology in both unstructured and structured P2P systems. In: International Conference on Parallel Processing, pp. 30–30. IEEE Computer Society, Xian (2007)
16. Liu, Y., Zhuang, Z., Xiao, L., Ni, L.M.: AOTO: Adaptive overlay topology optimization in unstructured P2P systems. In: Global Telecommunications Conference, pp. 4186–4190. IEEE Computer Society, San Francisco (2003)
17. Leitao, J., Marques, J.P., Pereira, J., Rodrigues, L.: X-bot: a protocol for resilient optimization of unstructured overlays. In: Proceedings of the 28th IEEE International Symposium on Reliable Distributed Systems, pp. 236–245. IEEE Computer Society, Niagara Falls (2009)
18. Dumitrescu, M., Andonie, R.: Clustering superpeers in p2p networks by growing neural gas. In: Proceedings of the 20th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, pp. 311–318. IEEE Computer Society, Munich (2012)
19. Babaoglu, O., Meling, H., Montresor, A.: Anthill: a framework for the development of agent-based peer-to-peer systems. In: Proceedings of the 22nd International Conference on Distributed Computing Systems, pp. 15–22. IEEE Computer Society, Vienna (2002)
20. Forestiero, A., Leonardi, E., Mastroianni, C., Meo, M.: Self-Chord: a bio-inspired p2p framework for self-organizing distributed systems. IEEE Trans. Netw **18**, 1651–1664 (2010)
21. Domic, N.G., Goles, E., Rica, S.: Dynamics and complexity of the schelling segregation model. Phys. Rev. E **83**, 96–111 (2011)
22. Thathachar, M., Sastry, P.S.: Networks of Learning Automata: Techniques for Online Stochastic Optimization. Kluwer Academic Publishers, Dordrecht (2004)
23. Lalitha, B., Rao, C.D.S.: GPS based topology matching algorithm for p2p systems. Int. J. Adv. Res. Comput. Sci. Softw. Eng **3**, 1–10 (2013)
24. Lalitha, B., Subba Rao, C.H.D.: Mitigation of topology mismatch problem based on node network positioning in unstructured p2p networks. i-manager's J. Comput. Sci **1**, 22–32 (2013)
25. Aggarwal, V., Feldmann, A., Scheideler, C.: Can ISPs and p2p users cooperate for improved performance? ACM SIGCOMM Comput. Commun. Rev **37**, 29–40 (2007)
26. Ratnasamy, S., Handley, M., Karp, R., Shenker, S.: Topologically-aware overlay construction and server selection. In: Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies, pp. 1190–1199. IEEE Computer Society, New York (2002)
27. Tian, R., Xiong, Y., Zhang, Q., Li, B., Zhao, B.Y., Li, X.: Hybrid overlay structure based on random walks. In: Peer-to-Peer Systems IV, pp. 152–162. Springer (2005)
28. Zhang, X.Y., Zhang, Q., Zhang, Z., Song, G., Zhu, W.: A construction of locality-aware overlay network: mOverlay and its performance. IEEE J. Sel. Areas Commun **22**, 18–28 (2004)
29. Scheidegger, M., Braun, T.: Improved locality-aware grouping in overlay networks. In: Kommunikation in Verteilten Systemen, pp. 27–38. Springer, Bern (2007)
30. Wolf, S., Merz, P.: Evolutionary local search for the super-peer selection problem and the p-hub median problem. In: Proceedings of the 4th International Conference on Hybrid Metaheuristics, pp. 1–15. Springer, Berlin, Heidelberg (2007)
31. Ju, H.-J., Du, L.-J.: Nodes clustering method in large-scale network. In: Proceedings of the 8th International Conference on Wireless Communications, Networking and Mobile Computing, pp. 1–4. IEEE Computer Society, Shanghai (2012)
32. Li, Y., Yu, Z.: An improved genetic algorithm for network nodes clustering. In: Proceedings of the Second International Conference on Information Computing and Applications, pp. 399–406. Springer Berlin Heidelberg, Qinhuangdao (2011)
33. Jiang, Y., You, J., He, X.: A particle swarm based network hosts clustering algorithm for peer-to-peer networks. In: International Conference on Computational Intelligence and Security, pp. 1176–1179. IEEE Computer Society, Guangzhou (2006)
34. Rostami, H., Habibi, J.: A mathematical foundation for topology awareness of p2p overlay networks. In: Grid and Cooperative Computing, pp. 906–918. Springer, Heidelberg (2005)
35. Liu, Y., Xiao, L., Liu, X., Ni, L.M., Zhang, X.: Location awareness in unstructured peer-to-peer systems. IEEE Trans. Parallel Distrib. Syst. **16**, 163–174 (2005)

36. Hsiao, H.C., Liao, H., Huang, C.C.: Resolving the topology mismatch problem in unstructured peer-to-peer networks. IEEE Trans. Parallel Distrib. Syst. **20**, 1668–1681 (2009)
37. Papadakis, H., Fragopoulou, P., Markatos, E., Roussopoulos, M.: ITA: innocuous topology awareness for unstructured p2p networks. IEEE Trans. Parallel Distrib. Syst. **24**, 1589–1601 (2013)
38. Xiao, L., Liu, Y., Ni, L.M.: Improving unstructured peer-to-peer systems by adaptive connection establishment. IEEE Trans. Comput. **54**, 1091–1103 (2005)
39. Yunhao, L., Xiao, L., Ni, L.M.: Building a scalable bipartite P2P overlay network. IEEE Trans. Parallel Distrib. Syst. **18**, 1296–1306 (2007)
40. Saghiri, A.M., Meybodi, M.R.: A distributed adaptive landmark clustering algorithm based on mOverlay and learning automata for topology mismatch problem in unstructured peer-to-peer networks. Int. J. Commun Syst (2015). doi:10.1002/dac.2977
41. Baumgart, I., Heep, B., Krause, S.: OverSim: A scalable and flexible overlay framework for simulation and real network applications. In: Peer-to-Peer Computing. pp, 87–88. IEEE Computer Society, Seattle, Washington (2009)
42. Baumgart, I., Heep, B., Krause, S.: OverSim: A flexible overlay network simulation framework. In: IEEE Global Internet Symposium, pp. 79–84. IEEE Computer Society, Anchorage (2007)
43. Baumgart, I., Gamer, T., Hübsch, C., Mayer, C.P.: Realistic underlays for overlay simulation. In: Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques, pp. 402–405. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), Brussels (2011)
44. Li, L., Alderson, D., Willinger, W., Doyle, J.: A first-principles approach to understanding the internet's router-level topology. ACM SIGCOMM Comput. Commun. Rev **34**, 3–14 (2004)
45. Ripeanu, M.: Peer-to-peer architecture case study: Gnutella network. In: First International Conference on Peer-to-Peer Computing, pp. 99–100. IEEE Computer Society, Linkoping (2001)
46. Liu, Y., Xiao, L., Esfahanian, A.H., Ni, L.M.: Approaching optimal peer-to-peer overlays. In: Proceedings of the 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, pp. 407–414. IEEE Computer Society, Atlanta (2005)
47. SNAP: Network datasets: Gnutella peer-to-peer network, http://snap.stanford.edu/data/p2p-Gnutella04.html
48. Mahadevan, P., Krioukov, D., Fomenkov, M., Huffaker, B., Dimitropoulos, X., Vahdat, A.: Lessons from Three Views of the Internet Topology. University of California, San Diego (2005)
49. Huffaker, B., Plummer, D., Moore, D., Claffy, K.C.: Topology discovery by active probing. In: Proceedings of Symposium on Applications and the Internet Workshops, pp. 90–96., Washington (2002)
50. Narendra, K.S., Thathachar, M.A.L.: Learning Automata: An Introduction. Prentice-Hall, Englewood Cliffs (1989)

Ali Mohammad Saghiri received the B. Sc. and M. Sc. degrees in computer engineering in Iran, in 2008 and 2010, respectively. He is currently the Ph.D. student of computer engineering in AmirKabir University of Technology, Tehran, Iran. His research interests include distributed systems, artificial intelligence, and bio-inspired computation.

Mohammad Reza Meybodi received the B.S. and M.S. degrees in Economics from Shahid Beheshti University in Iran, in 1973 and 1977, respectively. He also received the M.S. and Ph.D. degree from Oklahoma University, USA, in 1980 and 1983, respectively in Computer Science. Currently, he is a full professor in Computer Engineering Department, Amirkabir University of Technology, Tehran, Iran. His research interests include learning systems, parallel algorithms, and soft computing.