International Journal of Uncertainty,
Fuzziness and Knowledge-Based Systems
Vol. 23, No. 1 (2015) 1–31

DOI: 10.1142/S0218488515500014

Finding Maximum Clique in Stochastic Graphs Using Distributed Learning Automata

Alireza Rezvanian* and Mohammad Reza Meybodi†
Soft Computing Laboratory, Computer Engineering and Information Technology Department, Amirkabir University of Technology (Tehran Polytechnics), 424, Hafez Ave, Tehran, Iran
*a.rezvanian@aut.ac.ir
†mmeybodi@aut.c.ir

Received 16 March 2014
Revised 15 July 2014

Because of unpredictable, uncertain and time-varying nature of real networks it seems that stochastic graphs, in which weights associated to the edges are random variables, may be a better candidate as a graph model for real world networks. Once the graph model is chosen to be a stochastic graph, every feature of the graph such as path, clique, spanning tree and dominating set, to mention a few, should be treated as a stochastic feature. For example, choosing stochastic graph as the graph model of an online social network and defining community structure in terms of clique, and the associations among the individuals within the community as random variables, the concept of stochastic clique may be used to study community structure properties. In this paper maximum clique in stochastic graph is first defined and then several learning automata-based algorithms are proposed for solving maximum clique problem in stochastic graph where the probability distribution functions of the weights associated with the edges of the graph are unknown. It is shown that by a proper choice of the parameters of the proposed algorithms, one can make the probability of finding maximum clique in stochastic graph as close to unity as possible. Experimental results show that the proposed algorithms significantly reduce the number of samples needed to be taken from the edges of the stochastic graph as compared to the number of samples needed by standard sampling method at a given confidence level.

Keywords: Clique problem; maximum clique; stochastic graph, distributed learning automata; social networks.

1. Introduction

Several deterministic graph models such as Erdős-Rényi model,[1] Watts-Strogatz model[2] and Barabási-Albert model[3] are presented in the literature to model common properties of real networks such as small-world, scale free, and modular properties. Dynamic nature of many decision-making problems[4–8] in real world networks such as computer networks,[9] biochemical reaction networks,[10] and social networks[11–13] cause some of their structural and behavioral parameters be time varying parameters and for this reason deterministic graphs for modeling such networks may not be appropriate. In most scenarios of networking science in literatures, it is assumed that the weights associated with the edges of model graph are fixed. Such an assumption is not valid when the tasks/activities on networks vary with time. For example in online social networks the behaviors of users

such as friendship or behavior of taking comment on a post in *Facebook* vary over time with unknown probabilities.[14] Modeling social networks with deterministic graphs cannot take into consideration the continuum of activities of the users occurring over time. Even modeling social networks with weighted graphs in which the edge weights are assumed to be fixed can consider only a snapshot of the real network. In online social networks in addition to understanding the size and the structure of the community, the degree of association among the individuals within the community is significant for social network analysis. Understanding the characteristics of social relationship between users is the basis for capturing the realistic analysis of online social networks.

Community structure is one of the key interesting features of many real world networks and a variety of definitions are presented for community structure, among them definition of community in terms of clique is the most intuitive way. In a community in a network, we may have two types of links known as internal (strong tie) and external links (weak tie) [15]. Weak tie activities are important between communities, while the strong tie activities are important within communities. Deterministic view about the strength of ties only focuses on the impact of links and do not consider the time varying nature of such ties. For this reason, the standard definition of some concepts such as clique for defining community is too restrictive. For example, in online social networks, in addition to the size and the structure of the community the degree of association among the individuals intra/inter the communities play an important role and may vary over time. The strength of a community as a leader of communities which may not have necessarily the maximum number of members, because of its higher weight impresses on the weaker communities to propagate certain information.

Based on the discussion presented in previous paragraphs, it seems that stochastic graphs[16–18] in which weights associated to the edges are random variables is a better candidate as a graph model for real world networks with unpredictable and time varying nature. Once the graph model is chosen to be a stochastic graph, every feature of the graph such as path, clique, spanning tree and dominating set, to mention a few, should be treated as stochastic features. For example, choosing stochastic graph as the graph model of an online social network and defining community structure in terms of clique, and the associations among the individuals within the community as random variables, the concept of stochastic clique may be used to study community structure properties.

In this paper, maximum clique in stochastic graph is first defined and then several learning automata-based algorithms are proposed for solving maximum clique problem in stochastic graphs under unknown environment (that is, the probability distribution functions of the weights associated with the graph edges are unknown). The proposed algorithms need to take samples from the edges of the stochastic graph in order to find the clique with maximum expected weight. The proposed algorithms use a kind of guided sampling implemented with the aid of learning automata in order to reduce the number of samples needed to be taken from the edges of the stochastic graph for finding the clique with maximum expected weight. This is done by taking samples from the edges which are along the paths toward the stochastic maximum clique or edges of stochastic maximum clique itself. In the proposed algorithms, each vertex of the graph is equipped with a learning automaton whose actions correspond to choosing the edges of the corresponding vertex. The set of learning automata which forms a distributed learning automata tries to guide the sampling process in such a way that to obtain the clique with maximum expected weight with as fewer number of samples taken from the edges of the graph as possible. The guided sampling process implemented by distributed learning

automata tends to take more samples from the edges along the paths toward the clique with maximum expected weight or the clique itself instead of wandering around and taking unnecessary samples from non-promising edges.

To investigate the performance of the proposed algorithms, several experiments on different stochastic graphs are conducted. Experimental results show that the proposed algorithms significantly outperform the standard sampling method in terms of total number of samples needed to be taken from the graph in order to find the clique with maximum expected weight. It is also shown that by a proper choice of the parameters of the proposed algorithms, the probability of finding the MWC problem is as close to unity as possible. The rest of this paper is organized as follows. In Sec. 2, some preliminaries for the research work presented in this paper are given. Learning automata and some of their variants are briefly introduced in Sec. 3. The proposed algorithms for finding maximum clique in stochastic graph are described in Secs. 4 and 5 gives the simulation results. Section 6 concludes the paper.

2. Preliminaries

In this section, in order to provide background information for the remainder of the paper, we present a brief overview of clique problem and its variations, stochastic graph, stochastic clique which is introduced for the first time in this paper, learning automata, variable action set learning automata, and distributed learning automata.

2.1. *Clique*

Let $G = \langle V, E \rangle$ be a connected and undirected graph, where V is the set of vertices, and $E \in V \times V$ denotes the edge-set. A clique is a subset of vertices that are pair-wise adjacent in the graph. Maximum clique (MC) of graph G is a clique with maximum cardinality among all possible cliques of graph G. Maximum weight clique (MWC) problem is an important generalization of MC problem defined for weighted graph. MWC problem is the problem of finding a clique with the largest total weight. MC problem and MWC problem are both known to be NP-hard[19] and for this reason several approximation algorithms have been reported in the literature for solving them. The MC and MWC problems are applied in many areas of science.[20] For example MC or MWC of users in online social networks indicates a highly connected group of users as community.

Several studies are reported in the literatures for finding maximum clique in deterministic graph.[20] *Gibbons et al.*[21] solved MC problem by minimizing a quadratic form over the standard simplex via formulating MWC problem as a standard quadratic optimization problem. A new framework for MWC problem by linear complementary problem is presented in Ref. 22. They showed that generically, all stationary points of the MWC problem quadratic program represent strict complementary. Also branch and bound based algorithms are presented in for solving the MC problem to identify all maximum cliques in a finite graph.[23] An efficient branch and bound algorithm proposed by Babel,[24] which used upper and lower bounds according to coloring the weight of graph. A nice review on branch and bound algorithms for the MCP is given in Ref. 25 by Carmo *et al.* They implemented and compared eight different branch and bound algorithms under a unifying framework.

Since MC problem is an NP-hard problem, researchers have focused their efforts on finding heuristic algorithms for solving MC problem. Balas *et al.*[26] developed a genetic

algorithm with optimized crossover for the MC and Brunato *et al.*[27] proposed an evolutionary algorithm with guided mutation, a novel reactive and evolutionary algorithm (R-EVO) for solving MC problem. The capabilities of ant colony optimization meta-heuristic for solving the MC problem are investigated by Salnon *et al.*[28] Geng *et al.* presented a simple simulated annealing algorithm and test it on DIMACS maximum clique instances.[29] A multistart tabu search technique proposed by Wu *et al.*[30] They integrated a constrained neighborhood, a dynamic tabu tenure mechanism and a long term memory restart strategy in their algorithm. Zhou *et al.* proposed a new immune genetic algorithm based on the clonal selection strategy and uniform design sampling in order to solve the MC problem.[31] Another direction for designing heuristic algorithms for MC problem focused on the benefit of integrating a local search procedure within a typical algorithm to improve the solution process. Battini and Protasi[32] proposed a reactive local search procedure. Katayama *et al.*[33] proposed a variable depth search based algorithm which adaptively moves in the feasible search space. Benlic *et al.*[34] presented a breakout local search (BLS) without any particular adaptation. The basic idea behind BLS is to employ local search to explore local optima and use adaptive diversity strategies to travel between local optima in the search space. Soleimani-Pouri *et al.*[35] presented an ACO based algorithm for solving Maximum clique problem for deterministic graph. In this algorithm particle swarm optimization algorithm is used as local search method.[36] Recently, cooperative and parallel local search is also developed for MC problem. Pullan *et al.*[37] proposed a cooperating local search as a parallelized hyper heuristic for MC problem and resented an application to desktop multi core computers. Bhattacharyya e*t al.*[38] addressed the fuzzy version of maximum clique problem in fuzzy graph and solve it with neural networks. MC problem has many practical applications in wide spread of areas such as machine vision,[39] online shopping recommendation,[40] data clustering,[41] timetabling[42] and wireless sensor networks.[43]

2.2. *Stochastic graph and stochastic clique*

An edge-weighted connected and undirected graph can be described by a triple $G = \langle V, E, W\rangle$ where $V = \{v_1, v_2, \ldots, v_n\}$ is the set of vertices, $E = \{e_1, e_2, \ldots, e_m\} \in V \times V$ is the edge-set and $W = \{w_1, w_2, \ldots, w_m\}$ is the set of weights associated with the edges of graph. Graph G is said to be stochastic if weight w_i associated with edge e_i of graph is a random variable. *Maximum stochastic clique* in a stochastic graph is a clique with maximum expected weight.

2.3. *Learning automata*

A learning automaton (LA)[44] refers to an abstract model which randomly selects an action out of its finite set of actions and performs it on an unknown random environment. The action is chosen at random based on a probability distribution kept over the action-set and at each instant the given action is served as the input to the random environment. Environment then evaluates the selected action and responds to the automaton with a reinforcement signal. Based on selected action, and received signal, the automaton updates its internal state and selects its next action. Generally, the goal of a learning automaton is to find the optimal action from the action-set so that the average penalty received from the environment is minimized.

The environment of learning automata can be described by a triple $E = \{\alpha, \beta, c\}$, where:

$\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$ is the set of inputs (action sets);
$\beta = \{\beta_1, \beta_2, \ldots, \beta_m\}$ is the set of output values that can be taken by the reinforcement signal;
$c = \{c_1, c_2, \ldots, c_m\}$ is the set of penalty probabilities measured by the reaction of the environment; c_i is the penalty probability for action α_i.

The environment is classified into stationary or non-stationary by which the penalty probabilities are constant or variable respectively. According to the nature of the reinforcement signal β, the environment can be categorized into *P-model*, *Q-model* and *S-model*. P-model refers to an environment where in the reinforcement the reinforcement signal is able to take two binary values 0 and 1. The environment in which the reinforcement signal can take a finite number of the values in the interval [0, 1] is Q-model. In an S-model of the environment, the reinforcement signal is a continuous random variable in the interval [0, 1]. The random environment is said to be a non-stationary environment, if the penalty probabilities vary over time, and is called stationary otherwise. Figure 1 depicts the relationship between the learning automaton and its random environment.

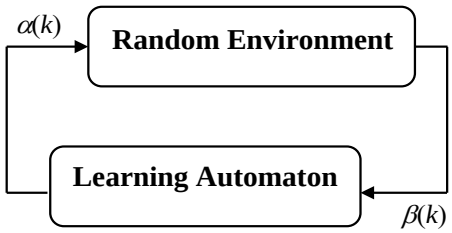


Fig. 1. Relationship between the learning automaton and its random environment.

Learning automata can be classified into two classes: fixed structure learning automata and variable structure learning automata.[44] A Variable-Structure Learning Automaton (VSLA) is defined by a quadruple $\langle p, \alpha, \beta, T \rangle$, where $p(k + 1) = T[p(k), \alpha(k), \beta(k)]$ is the reinforcement scheme of automaton (also known as learning algorithm), $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$ is the action sets, $\beta = \{\beta_1, \beta_2, \ldots, \beta_r\}$ denotes the set of inputs and $p = \{p_1, p_2, \ldots, p_r\}$ is the probability of each action. The learning algorithm refers to a recurrence relation which is used to modify the action probability vector. Consider $\alpha_i(k) \in \alpha$ denotes the action that is chosen by a learning automaton and $p(k)$ is the probability vector defined over the set of action at instant k. Let a denotes reward parameter which determines the amount of increase of the action probability values and consider b is the penalty parameter determining the amount of decrease of the action probabilities values. Let r is the number of actions that learning automaton can take. At each instant k, the action probability vector $p(k)$ is updated by the linear learning algorithm given in Eq. (1), on the other hand, if the chosen action $\alpha_i(k)$ is rewarded by the random environment, and it is updated according to Eq. (2), if the taken action is penalized.

If $a = b$, the recurrence equations (1) and (2) are called linear reward-penalty ($L_{R\text{-}P}$) algorithm; if $a >> b$, they are called linear reward-ε-penalty ($L_{R\text{-}\varepsilon P}$) algorithm; and finally

if $b = 0$, they are called linear reward-Inaction (L_{R-I}) algorithm. In L_{R-I}, the action probability vectors remain unchanged when the taken action is penalized by the environment.

$$p_j(k+1) = \begin{cases} p_j(k) + a[1 - p_j(k)] & j = i \\ (1-a)p_j(k) & \forall j \neq i \end{cases} \quad (1)$$

$$p_j(k+1) = \begin{cases} 1 - bp_j(k) & j = i \\ \left(\frac{b}{r-1}\right) + (1-b)p_j(k) & \forall j \neq i \end{cases} \quad (2)$$

Learning automata have been used as optimization tools in the complex and dynamic environments where a large amount of uncertainty or lacking the information about the environment exists. In the recent years, learning automata have been successfully applied to a wide variety of applications such as graph problems,[45,46] pattern recognition,[47] large scale optimization,[48] cloud computing,[49] cellular networks[50] and complex networks.[51]

2.4. *Variable action set learning automata*

A variable action set learning automaton is an automaton in which the number of actions available at each instant changes with time.[52] Such an automaton includes finite set of n actions, $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$. $A = \{A_1, A_2, \ldots, A_m\}$ is the set of action subsets and $A(k) \subseteq \alpha$ denotes the subset of all the actions can be selected by the learning automaton, at each instant k. According to the probability distribution $q(k) = \{q_1(k), q_2(k), q_m(k)\}$, an external agency chooses randomly the particular action subsets, where: $q_i(k) = prob[A(k) = A_i | A_i \in A, 1 \leq i \leq 2^n - 1]$. $\hat{p}_i(k) = prob[\alpha(k) = \alpha_i \mid A(k), \alpha_i \in A(k)]$ is the probability of choosing action α_i, if the action subset $A(k)$ has already been selected and also $\alpha_i \in A(k)$. The scaled probability $\hat{p}_i(k)$ is defined as

$$\hat{p}_i(k) = \frac{p_i(k)}{K(k)} \quad (3)$$

where $K(k) = \sum_{\alpha_i \in A_k} p_i(k)$ is the sum of the probabilities of the actions in subset $A(k)$, and $p_i(k) = prob[\alpha(k) = \alpha_i]$.

In a variable action set learning automaton, the procedure of selecting an action and updating the action probabilities can be described as follows. Let $A(k)$ is the action subset selected at instant k. Before selecting an action, the probabilities of all the actions in the selected subset are scaled using Eq. (3). Then, the automaton randomly selects one of its possible actions according to the scaled action probability vector $\hat{p}_i(k)$. The learning automaton updates its scaled action probability vector based on the response received from the environment. Note that in this step, the probability of the available actions is only updated. Finally, the probability vector of the actions of the selected subset is rescaled as follows for all $\alpha_i \in A(k)$.

$$p_i(k+1) = \hat{p}_i(k+1) \cdot K(k). \tag{4}$$

2.5. *Distributed learning automata*

A distributed learning automata (DLA)[16] shown in Fig. 2 is a network of interconnected learning automata which collectively cooperate to solve a particular problem. The number of actions for a particular LA in DLA is equal to the number of LA's that are connected to this LA. Selection of an action by a LA in DLA activates another LA which corresponds to this action. Formally, a DLA can be defined by a quadruple $\langle A, E, T, A_0 \rangle$, where $A = \{A_1, A_2, \ldots, A_n\}$ is the set of learning automata, $E \subset A \times A$ is the set of edges where edge (v_i, v_j) corresponds to action α_i^j of automaton A_i, T is the set of learning algorithms with which learning automata update their action probability vectors, and A_0 is the root automaton of DLA at which activation of DLA starts.

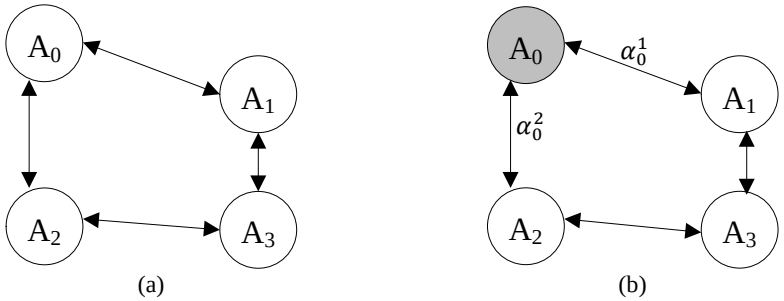


Fig. 2. Distributed learning automata.

The operation of a DLA can be described as follows: At first, the root automaton A_0 randomly chooses one of its outgoing edges (actions) according to its action probabilities and activates the learning automaton at the other end of the selected edge. The activated automaton also randomly selects an action which results in activation of another automaton. The process of choosing actions and activating automata is continued until a leaf automaton (an automaton which interacts with the environment) is reached. The chosen actions, along the path induced by the activated automata are applied to the random environment. The environment evaluates the applied actions and emits a reinforcement signal to DLA. The activated learning automata along the chosen path update their action probability vectors on the basis of the reinforcement signal according to the learning algorithms. The paths from the unique root automaton to one of the leaf automata are selected until the probability with which one of the chosen paths is close enough to unity. Each DLA has exactly one root automaton which is always activated, and at least one leaf automaton which is activated probabilistically. For example in Fig. 2, every automaton has two actions. If automaton A_0 selects α_0^2 from its action set, then it will activate automaton A_2. Afterward, automaton A_2 chooses one of its possible actions and so on. DLA has found successful applications in several arias including: wireless Ad Hoc networks,[53] grid computing[54] and complex networks.[51]

3. Proposed Algorithm

In this section, we propose a distributed learning automata based algorithm for finding the maximum weight clique (MWC) in the stochastic graph. It is assumed that the edge weight of graph is a positive random variable with unknown probability distribution function. The proposed algorithm tries to find a clique with the maximum expected weight by sampling the edges of the graph in such a way that the number of samples taken from the edges be minimized. The sampling from the edges will be guided by a distributed learning automata isomorphic to the input graph. Let $G = \langle V, E, W\rangle$ be the input stochastic graph, where $V = \{v_1, v_2, \ldots, v_n\}$ is the set of vertices, E is the edge set, and $W = \{w_1, w_2, \ldots, w_m\}$ is a set of random variables each of which is associated to one edge of the input graph. n and m be the total number of vertices and edges in the input graph respectively. The proposed algorithm uses a distributed learning automata which is isomorphic to the input graph. This network can be defined by 2-tuple $\langle A, \alpha\rangle$ where $A = \{A_1, A_2, \ldots, A_n\}$ is the set of learning automata each of which assigned to a vertex of the input graph; specifically A_i is assigned to v_i, $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ is the action-set in which $\alpha_i = \{\alpha_i^1, \alpha_i^2, \ldots, \alpha_i^{r_i}\}$ is the set of actions where r_i is the number actions that can be taken by learning automata A_i for each $\alpha_i \in \alpha$. An action of an automaton corresponds to choosing an edge of the corresponding vertex. At any time each learning automaton can be in one of two modes: active and inactive. Initially, all learning automata are in inactive mode.

The proposed algorithm consists of four phases: 1. *initialization*, 2. *finding a candidate stochastic clique*, 3. *calculating dynamic threshold*, and 4. *updating action probabilities*. After the initialization is performed the algorithm iterates phases 2, 3, and 4 until stopping conditions are met. In initialization phase learning automata $A = \{A_1, A_2, \ldots, A_n\}$ are created and assigned to the vertices. In phase two a candidate stochastic clique by performing a guided search in the graph with the help of learning automata residing in the vertices of the graph is determined. In phase 3, a threshold which is the average of the weights of all candidate stochastic cliques found in all iterations performed so far is computed and in phase 4, the action probability vectors of visited vertices during phase 1 are updated. Finally the algorithm stops if the stop conditions are met. The flowchart of the proposed algorithm is shown in Fig. 3 and details of each phase are described as follows.

3.1. *Initialization*

A distributed learning automata $\langle A, \alpha\rangle$ whose topology is isomorphic to the input graph is created. $A = \{A_1, A_2, \ldots, A_n\}$ is the set of learning automata and $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ where α_i is the action set for automaton A_i and $\alpha_i = \{\alpha_i^1, \alpha_i^2, \ldots, \alpha_i^{r_i}\}$ is the set of actions for learning automaton A_i. Learning automaton A_i which is associated to vertex v_i has r_i actions each of which corresponds to selecting one of the outgoing edges of vertex v_i. Let $p(v_i) = \{p_i^1, p_i^2, \ldots, p_i^{r_i}\}$ be the action probability vector of learning automaton A_i. Initially $p_i^j = 1/r_i$ for all j. Each learning automaton can be in either active or inactive mode. At the beginning of the algorithm all learning automata are in inactive mode. Set θ at any time during the execution of the clique finding algorithm contains the vertices of

the stochastic clique found up to that point. θ initially contains vertex v_i which is selected randomly. Weight of θ denoted by $\overline{w}(\theta)$ is defined to be the sum of the weights of the edges between every pair of vertices in θ.

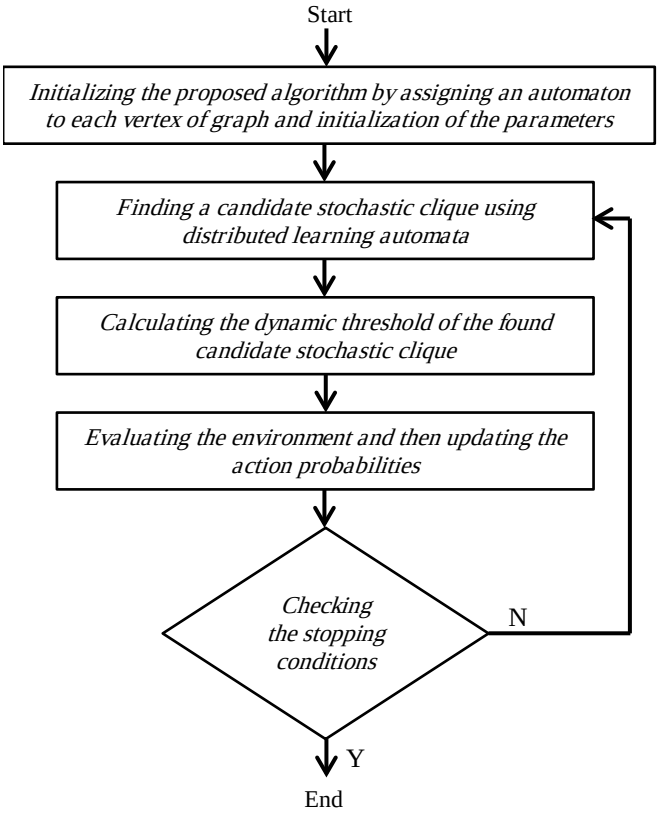


Fig. 3. Flowchart of the proposed Algorithm 1.

3.2. *Finding a candidate stochastic clique*

In this phase, one of the vertices of the input graph is randomly selected and added to θ_k. Then learning automaton associated to this vertex is activated. Let this vertex be v_i. Learning automaton A_i chooses one of its actions according to its action probability vector. Let α_i^j be the chosen action corresponding to edge (v_i, v_j). Each inactive learning automaton connected to A_i removes action α_j^i from its action-set and then scales its action probabilities according to Eq. (3). Then automaton A_j is activated and selects an action. If adding vertex v_j to θ_k forms a clique then vertex v_j will be added to the candidate stochastic clique θ_k. The process of activating a learning automaton, choosing an action, changing the number of actions for inactive automata connected to the activated chosen automaton and adding a proper vertex to candidate clique is repeated until we cannot

activate an automaton. We may not be able to activate an automaton if either the action of the activated automaton is empty or all the vertices of the graph have been visited. The reason for removing the action selected by the activated from the list of actions of automata connected to the activated automata is to avoid the formation of cycles during the traversal of the graph which may lead to a situation that the algorithm repeatedly traverses a part of the graph forever.

3.3. *Calculating dynamic threshold*

The *dynamic threshold* at iteration *k*, *DT*(*k*), which is the average of the weights of all stochastic cliques found so far is computed according to Eq. (5).

$$DT(k) = \frac{1}{k}\sum_{i=1}^{k} \bar{w}(\theta_i) \tag{5}$$

where *k* is the number of iteration, $\bar{w}(\theta_i)$ is the sum of the weights of the edges between every pair of vertices in θ_i. Dynamic threshold is used to decide whether or not a clique found during an iteration can becomes a candidate for stochastic maximum clique. If the clique found during iteration has a weight greater than the dynamic threshold *DT*(*k*) then that clique becomes a candidate for the stochastic maximum clique.

3.4. *Updating action probabilities*

In this phase, the probability vectors of learning automata assigned to the visited vertices of the graph are updated according to the learning algorithm on the basis of the result obtained from the previous phase. If $\bar{w}(\theta_k)$ is greater than or equal to the dynamic threshold *DT*(*k*), then the actions selected by all learning automata along the traversed path are rewarded and penalized otherwise. At the end of this phase, the actions removed from the set of actions of a learning automaton will be added to the action set of that learning automaton and then its action probability is rescaled according to Eq. (4) to reflect this addition. Also, all activated learning automata will become inactive.

3.5. *Stopping conditions*

The algorithm stops when the number of iterations *k* exceeds a predefined number K_{max} or *PC*(*k*) defined by equation (6) reaches a predefined value P_{max}.

$$PC(k) = \prod_{v_i \in \theta_k} \left(max(p(v_i))\right) \tag{6}$$

PC(*k*) is the product of maximum probabilities in probability vectors of learning automata of the vertices of candidate stochastic clique θ_k at iteration *k*. $p(v_i)$ is the action probability vector of learning automaton A_i residing in vertex v_i.

Figure 4 shows the pseudo code for the proposed Algorithm 1.

Algorithm 1: the proposed algorithm for finding maximum stochastic clique

Input: Stochastic Graph $G\langle V, E, W\rangle$, Thresholds P_{max}, K_{max}
Output: The stochastic maximum clique
Assumptions:
Assign an automaton A_i to each vertex v_i and initially set to the inactive mode;
Let k denotes the iteration number which is initially set to 1;
Let θ_k denotes the candidate stochastic maximum clique at iteration k;
Begin
Let *DT(k)* denotes the dynamic threshold at iteration k initially set to 0;
Let *PC(k)* is the product of the probabilities the edges of candidate stochastic maximum clique θ_k;
Repeat
Let $\overline{w}(\theta_k)$ is the average weight of all samples taken from θ_k;
v_i is randomly chosen as the initial candidate stochastic clique θ_k;
While (able to activate automaton) **Do**
Automaton A_i is activated and then chooses an action using its action probability vector;
Let the action chosen by A_i be (v_i , v_j);
Visit (sample) the chosen edge (v_i , v_j);
If adding vertex v_j to θ_k forms a clique **then**
Add vertex v_j to the candidate stochastic clique θ_k;
Compute the weight for the new clique $\overline{w}(\theta_k)$;
End if
Remove action α_j^i from action set of each inactive learning automaton connected to A_i and scale their action probabilities according to equation (3);
$v_i \leftarrow v_j$;
End While
Calculate dynamic threshold *DT(k)* using equation (5);
If $(\overline{w}(\theta_k) \geq DT(k))$ **Then**
Reward the chosen actions by all the activated learning automata;
Else
Penalize the chosen actions by all the activated learning automata;
End If
Calculate *PC(k)* using equation (6);
$k \leftarrow k + 1$;
Insert actions of all learning automata and rescale their action probabilities;
Deactivate all learning automata;
Until (*PC*(k) is greater than P_{max} **or** k is greater than K_{max})
End Algorithm

Fig. 4. Pseudo code for proposed Algorithm 1.

3.6. *Improvements*

The performance of any learning automata based algorithm is very sensitive to learning rate of the learning algorithm. Large value for learning rate results in increasing the speed of convergence and decreasing the accuracy of the algorithm, while small value for learning rate results in increasing the accuracy and decreasing the speed of convergence of the algorithm. In Algorithm 1, all learning automata use a same learning rate; it means that the algorithm considers a same degree of importance for the edges being traversed during the traversal of the graph. In the remaining part of this section several strategies for adjusting the learning rate are considered. These strategies try to adjust the learning rates in such a way that higher degree of importance be given to the edges leading to a traversal of the graph which produces better solution and higher rate of convergence.

3.6.1. *Strategy 1*

According to Algorithm 1, the probability of choosing edges which are more promising (belonging to the maximum stochastic clique or are on the paths to the maximum stochastic clique) gradually increases and for other edges decrease. It seems one way to speed up the learning process (which may leads to taking fewer numbers of samples taken from the graph) is to use higher learning rate for those learning automata in the vertices which are along more promising paths to the stochastic maximum clique or in the vertices which are part of the stochastic clique. To achieve this, we may use the Eq. (7) as the learning parameter $a_i(k)$ of automaton A_i at iteration k.

$$a_i(k) = c \cdot p_i^j(k) \tag{7}$$

where c is a value between 0 and 1. $p_i^j(k)$ is the action probability of choosing edge (v_i, v_j). The proposed Algorithm 1 in which the learning rate is adjusted according to strategy 1 is called *Algorithm 2*.

3.6.2. *Strategy 2*

In this strategy learning parameter a_i for automaton A_i at iteration k is determined according to the following equation:

$$a_i(k) = c \cdot d_i^j(k) \tag{8}$$

where c is a value between 0 and 1, $d_i^j(k)$ is an estimation of reward probability for edge (v_i, v_j) computed by the algorithm at iteration k. $d_i^j(k)$ is computed by dividing the number of times that action α_i^j is selected up to time k by automaton A_i and received reward by the number of times that automaton A_i is activated. In this way promising paths toward the maximum stochastic will be traversed. Algorithm 1 in which the learning rate is adjusted according to strategy 2 is called *Algorithm 3*.

3.6.3. *Strategy 3*

In this strategy learning parameter $a_i(k)$ for automaton A_i at iteration k is determined according to the following equation

$$a_i(k) = c \cdot (p_i^j(k))^{(K_{max}-k)/K_{max}} \tag{9}$$

where c is a value between 0 and 1, $p_i^j(k)$ is the probability of choosing action α_i^j which corresponds to choosing edge (v_i, v_j) at iteration k and K_{max} is the maximum number of iterations which the algorithm is allowed to continue. This strategy is similar to strategy 1 except that in this strategy the learning rate increases more smoothly. Algorithm 1 in which the learning rate is adjusted according to strategy 3 is called *Algorithm 4*.

3.6.4. *Strategy 4*

In all stochastic search techniques, in order to establish a balance between exploration and exploitation process, the step length of the algorithm gradually decreases. This causes

that the algorithm considers a higher exploration with lower exploitation for initial periods of the algorithm while have a higher exploitation with lower exploration for final periods of the algorithm. In strategy 4, we let learning rate a_i used by automaton A_i at iteration k be computed according to the following equation:

$$a_i(k) = \frac{(K_{max} - k)((a_{min} - a_{max}) + a_{max})}{K_{max}} \tag{10}$$

where K_{max} is the maximum number of iterations, a_{min} and a_{max} are minimum and maximum values that the learning rate can have and set in the algorithm respectively. Using this strategy it is expected that the proposed algorithm searches more regions of the graph (exploration) during the early periods of the search process and focuses on the regions near the solution during the final periods of the search process. The proposed Algorithm 1 in which the learning rate is adjusted on the basis of strategy 4 is called *Algorithm 5*.

4. Experimental Results

In this section, performance of the proposed algorithms is investigated on a number of synthetic stochastic graphs.

Tables 1 and 2 describe the stochastic graphs are used for the experimentations and their characteristics.

Table 1. Number of cliques of different sizes in test graphs.

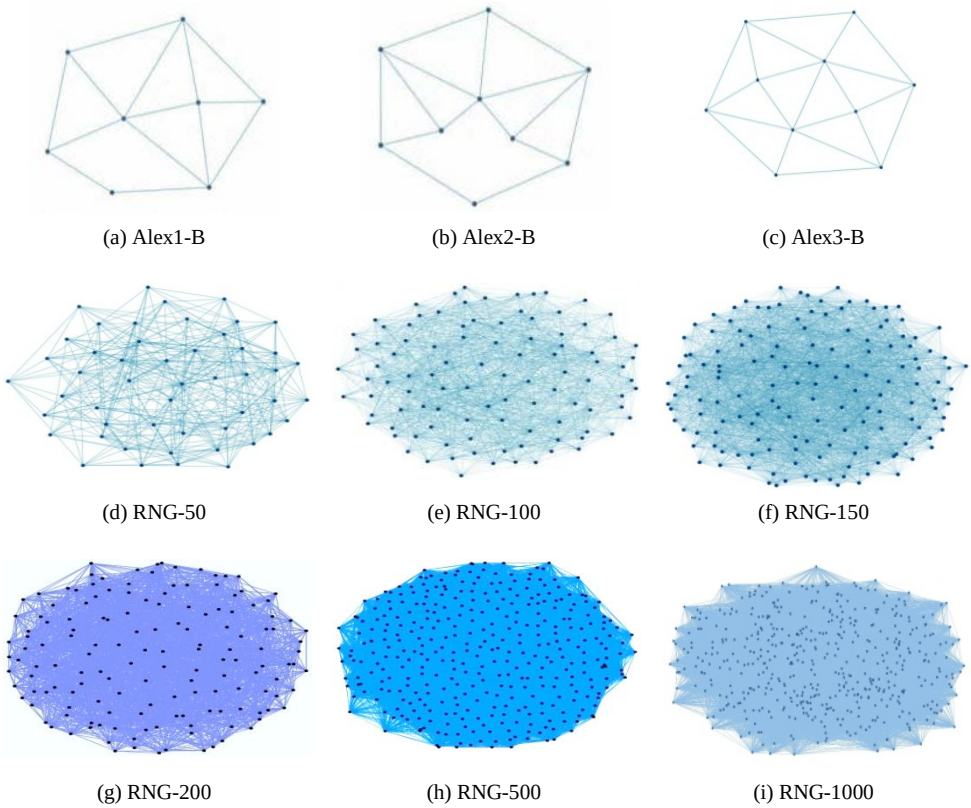
Graphs	Number of cliques of size 2	Number of cliques of size 3	Number of cliques of size 4	Number of cliques of size 5	Number of cliques of size 6	Number of cliques of size 7	Number of cliques of size 8	Total number of cliques
Alex1-B	14	6	0	0	0	0	0	20
Alex2-B	15	6	0	0	0	0	0	21
Alex3-B	21	12	0	0	0	0	0	33
RNG-50	240	105	7	0	0	0	0	352
RNG-100	998	607	232	7	0	0	0	1844
RNG-150	2172	1454	809	30	0	0	0	4465
RNG-200	3967	2065	3026	299	1	0	0	9358
RNG-500	24853	3060	72941	22368	691	2	0	123915
RNG-1000	99895	4152	119867	47685	1918	278	12	273807

Table 1 gives the number of cliques of different sizes for each graph and Table 2 gives other characteristics such as number of vertices, number of edges, number of cliques, vertices of the clique with maximum expected weight, size of the maximum clique, and expected weight of maximum clique. The first three graphs are well-known stochastic graphs which are borrowed from Ref. 55, with their weights of edges being random variables. The other stochastic graphs are generated randomly. These graphs have n vertices (*i.e.*, $n \in \{10-100, 150, 200, 500, 1000\}$) where each pair of vertices are connected with probability $p \in \{0.2, 0.3\}$ and edge weights for RNG-50 and RNG-100 are random variables with exponential distributions with mean $\lambda = 1$ and for RNG-200, RNG-500, and RNG-1000 are also random variables with exponential distributions whose

mean are chosen randomly from set $\lambda \in \{1,2,3,4,5\}$. The test graphs used for the experimentations are given in Fig. 5.

Table 2. Stochastic graphs used in the simulations.

Graphs	Parameter of distribution	Number of vertices	Number of edges	Vertices of the clique with maximum expected weight	Size of the maximum clique	Expected weight of maximum clique
Alex1-B	-	8	14	{6,7,8}	3	51.06
Alex2-B	-	9	15	{4,5,6}	3	56.97
Alex3-B	-	10	21	{3,6,7}	3	41.26
RNG-50	λ=1	50	240	{17,20,27,48}	4	6.81
RNG-100	λ=1	100	998	{36,53,62,77,97}	5	13.56
RNG-150	λ=1	150	2172	{10,36,48,61,83}	5	13.73
RNG-200	λ∈{1,2,3,4,5}	200	3967	{1,79,94,121,153,160}	6	31.69
RNG-500	λ∈{1,2,3,4,5}	500	24853	{16,93,173,259,343,350,369}	7	62.10
RNG-1000	λ∈{1,2,3,4,5}	1000	99895	{19,124,202,391,425,616,658,869}	8	75.56



(a) Alex1-B (b) Alex2-B (c) Alex3-B

(d) RNG-50 (e) RNG-100 (f) RNG-150

(g) RNG-200 (h) RNG-500 (i) RNG-1000

Fig. 5. Visualization of stochastic graphs for experiments.

4.1. *Experimental settings*

To show the performance of the proposed algorithm, several experiments are conducted on the stochastic graphs described in Tables 1 and 2. All simulations are carried out 30 times and their averages of each parameter are reported for the experiments. The stopping conditions of proposed algorithm is either the number of iteration reached 100,000 iterations or the value of $PC(k)$ which mentioned as Eq. (6) for product of maximum probabilities in probability vectors of learning automata of the vertices of candidate stochastic clique θ_k at iteration k is greater than 0.9. All experiments for the proposed algorithms are evaluated in terms of total number of samples taken from graph (TNS) and percentage of runs converged to the maximum stochastic clique (PCR). In reported tables, results for the best algorithm is shown in bold. For all algorithms, the reinforcement scheme used for updating the action probability vector is L_{R-I}. For Algorithm 1, the learning rate a is set to 0.1 and for Algorithm 2 to 4, the parameter c is set to 0.3. For Algorithm 5, a_{min} and a_{max} are set to 0.01 and 0.1, respectively.

4.2.1. *Experiment I*

This experiment is conducted to study the behavior of Algorithm 1 during the process of finding the solution. In order to perform this experiment, we plot both $PC(k)$ and $DT(k)$ versus iteration number. $DT(k)$ is scaled between (0,1) by dividing $DT(k)$ by the weight of maximum clique for each graph. Note that $DT(k)$, the dynamic threshold, is the average of the weights of all stochastic cliques found up to iteration k (Eq. (5)). The results of this experiment for different test graphs are given in Fig. 6. As it is shown, DT gradually converges to the weight of maximum clique and at the same time PC converges to 1. This means that the algorithm gradually finds the clique with maximum expected weight. Similar result can be obtained for other algorithms.

4.2.2. *Experiment II*

This experiment is carried out to study the performance of the proposed algorithms (Algorithm 1 and its variants) in terms of total number of samples taken from graph (TNS) and the percentage of runs converged to the maximum stochastic clique (PRC). For Algorithm 1 initial value for learning rate (a) is set to 0.1 and for other algorithms in which the learning rate is adjusted adaptively, constant c is set to 0.3. The results are given in Fig. 7. From the results we may conclude the following:

- For large graphs (RNG-50, RNG-100, RNG-200, RNG-500 and RNG-1000) all adaptive strategies (Algorithms 2 to 5) require fewer samples to be taken from the graph than Algorithm 1 whereas for small graphs (Alex-1-B, Alex2-B, Alex3) Algorithm 1 requires fewer samples comparing to Algorithms 2 and Algorithms 5. The results show that Algorithm 1 for which learning rate is fixed works better for small graph in terms total number of samples taken.
- In terms of total number of samples and percentage of runs converged to the maximum stochastic clique Algorithm 3 which uses adaptive strategy 2 has the best performance for all test graphs.

- In terms of percentage of runs converged to the maximum stochastic clique, all the proposed algorithms perform very good on small graphs (Alex-1-B, Alex2-B, Alex3-B).
- For all algorithms increasing the size of graph results decreasing the percentage of runs converged to the maximum stochastic clique.

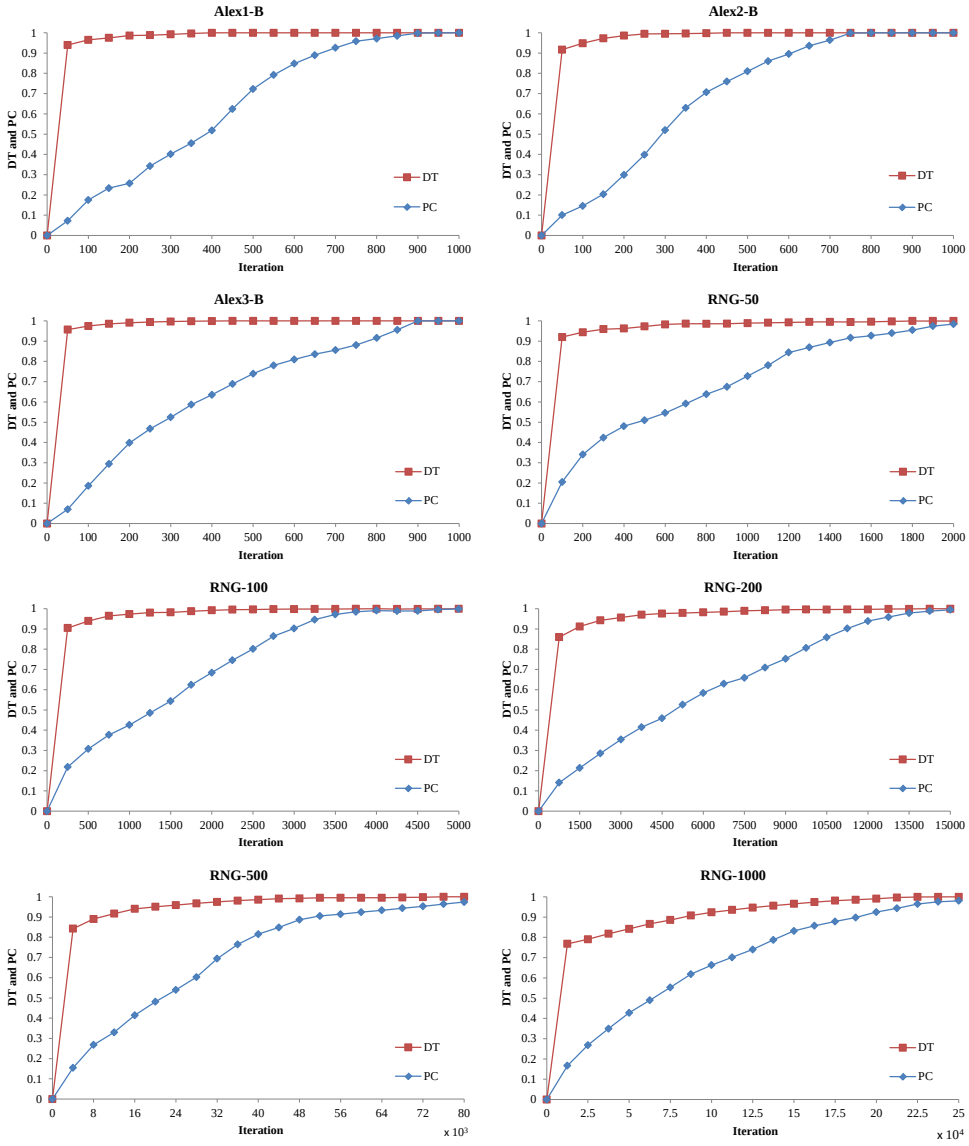
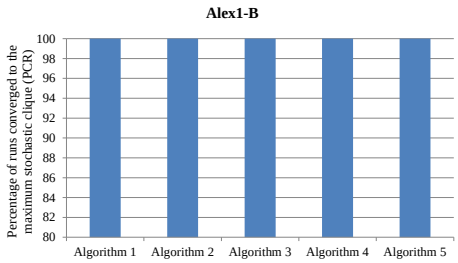
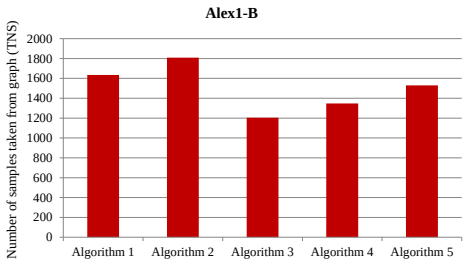
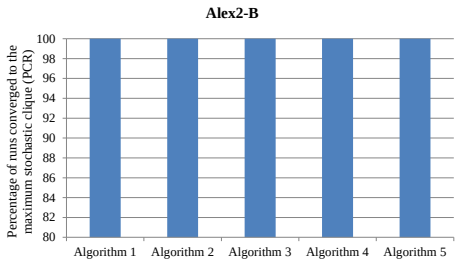
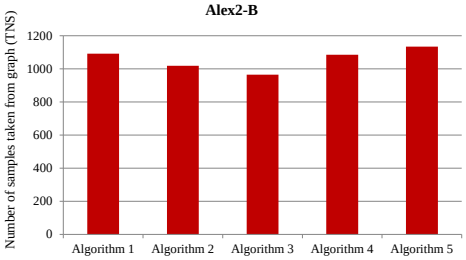


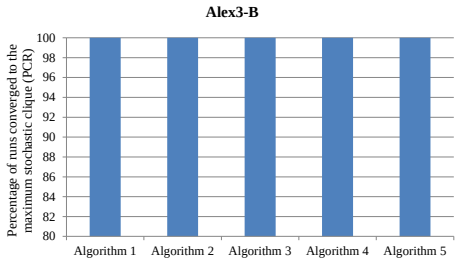
Fig. 6. The plot of *PC* and *DT* versus iteration number for different graphs.

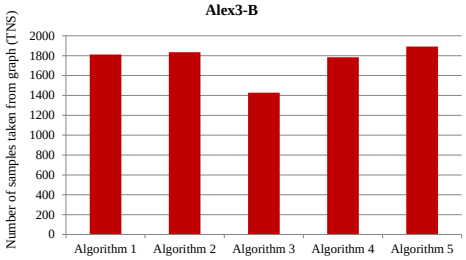

Alex1-B
Percentage of runs converged to the maximum stochastic clique (PCR)
100
98
96
94
92
90
88
86
84
82
80
Algorithm 1
Algorithm 2
Algorithm 3
Algorithm 4
Algorithm 5

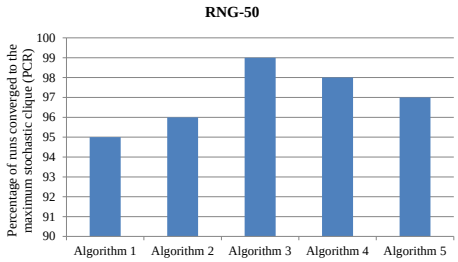

Alex1-B
Number of samples taken from graph (TNS)
2000
1800
1600
1400
1200
1000
800
600
400
200
0
Algorithm 1
Algorithm 2
Algorithm 3
Algorithm 4
Algorithm 5

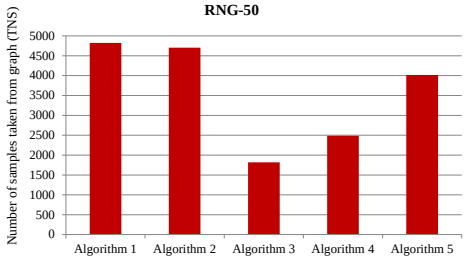

Alex2-B
Percentage of runs converged to the maximum stochastic clique (PCR)
100
98
96
94
92
90
88
86
84
82
80
Algorithm 1
Algorithm 2
Algorithm 3
Algorithm 4
Algorithm 5

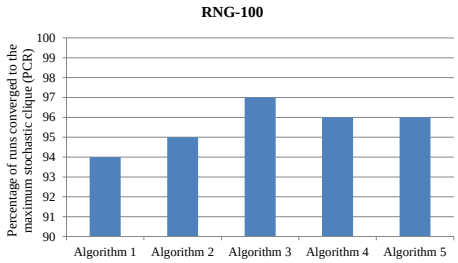

Alex2-B
Number of samples taken from graph (TNS)
1200
1000
800
600
400
200
0
Algorithm 1
Algorithm 2
Algorithm 3
Algorithm 4
Algorithm 5

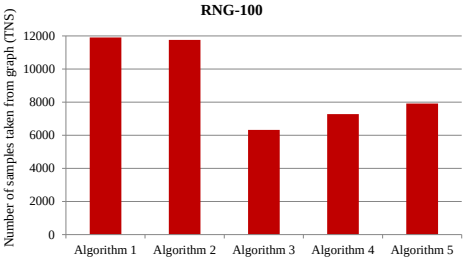

Alex3-B
Percentage of runs converged to the maximum stochastic clique (PCR)
100
98
96
94
92
90
88
86
84
82
80
Algorithm 1
Algorithm 2
Algorithm 3
Algorithm 4
Algorithm 5


Alex3-B
Number of samples taken from graph (TNS)
2000
1800
1600
1400
1200
1000
800
600
400
200
0
Algorithm 1
Algorithm 2
Algorithm 3
Algorithm 4
Algorithm 5


RNG-50
Percentage of runs converged to the maximum stochastic clique (PCR)
100
99
98
97
96
95
94
93
92
91
90
Algorithm 1
Algorithm 2
Algorithm 3
Algorithm 4
Algorithm 5


RNG-50
Number of samples taken from graph (TNS)
5000
4500
4000
3500
3000
2500
2000
1500
1000
500
0
Algorithm 1
Algorithm 2
Algorithm 3
Algorithm 4
Algorithm 5


RNG-100
Percentage of runs converged to the maximum stochastic clique (PCR)
100
99
98
97
96
95
94
93
92
91
90
Algorithm 1
Algorithm 2
Algorithm 3
Algorithm 4
Algorithm 5


RNG-100
Number of samples taken from graph (TNS)
12000
10000
8000
6000
4000
2000
0
Algorithm 1
Algorithm 2
Algorithm 3
Algorithm 4
Algorithm 5

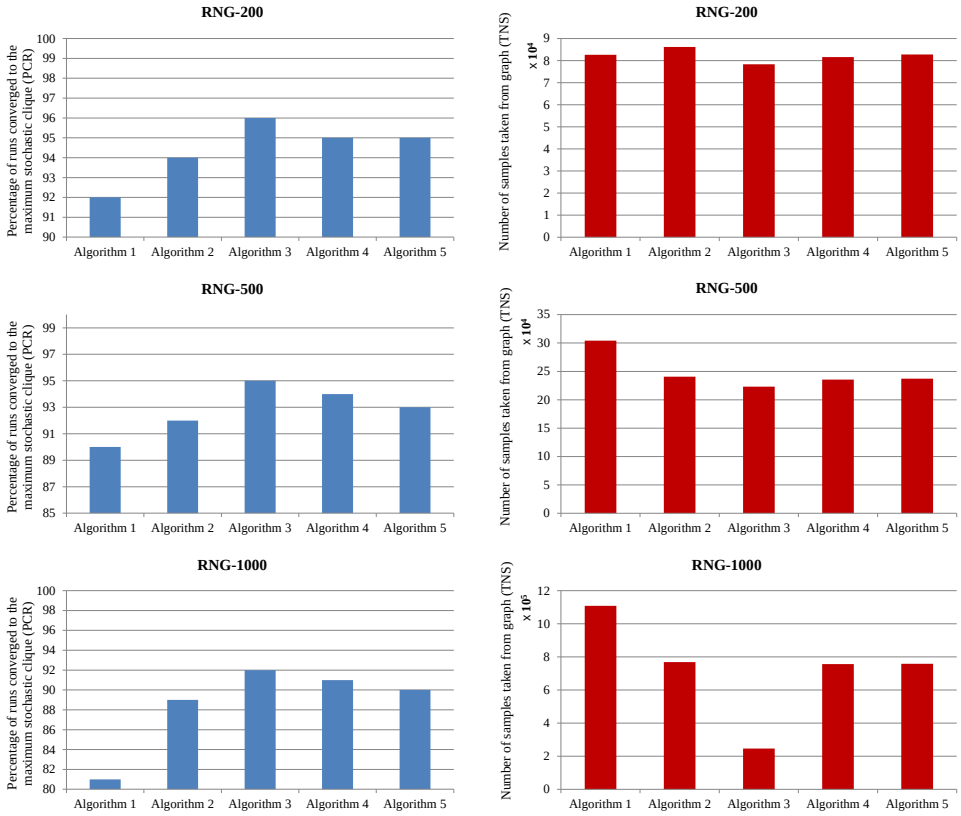


Fig. 7. Comparison of the proposed algorithms in terms of percentage of runs converged (PCR) to the maximum stochastic clique and the number of samples taken from graph (TNS).

4.2.3. *Experiment III*

In this experiment, we compare the number of samples needed by the proposed algorithms in order to achieve a given accuracy and then compare it with the number of samples required by standard sampling method (SSM) to reach the same accuracy. In order to a have fair comparison, both the proposed algorithm and SSM method must use the same confidence level $1-\varepsilon$. The proposed algorithm may reach the confidence level of $1-\varepsilon$ by a proper choice of learning rate a. According to Ref. 56, such a learning rate can be estimated using equation $ax/(e^{ax}-1) = \max_{i\neq j}(d_j/d_i)$ where d_i is the reward probability of action α_i. For a complete discussion about estimation of learning rate a the reader may refer to appendix I. Based on the SSM, to obtain a certain confidence level $1-\varepsilon$ for a subset of edges in graph, we need to build a confidence level $1-\varepsilon_i$ for each edge e_i such that $\sum_{i=1}^{n}\varepsilon_i = \varepsilon$. In this experiment, same confidence level $1-\varepsilon_0$ is assumed for all edges of the stochastic graph. The minimum required number of samples for each edge of graph for SSM is calculated subject to $p[|\bar{x}_n - \mu| < \delta] \geq 1-\varepsilon$, where $\delta = 0.001$.

For this experimentation, error rate ε varies from 0.4 to 0.1 with increment 0.1 (or convergence rate from 60% to 90%) for the proposed algorithms. Different graphs with size from 20 to 100 and some large case (200, 500, 1000) are used. Table 3 to Table 6 shows the results of this experimentation. In order to verify statistical differences among all the algorithms a statistical test is also performed over the algorithms. The statistical results of comparing algorithms by the two-tailed *t*-Test with 28 degrees of freedom at a 0.05 level of significance are given in Table 7 to compare each pair of results. In Table 7, the *t*-Test result regarding *Algorithm i* vs. *Algorithm j* is shown as "***s***+" and "***s***–" when *Algorithm i* is significantly better than, or significantly worse than *Algorithm j* respectively. From the results, several conclusions can be made:

Table 3. Number of samples taken from different test graphs using the proposed algorithms and using SSM at confidence level 0.6.

Graph size	Algorithm 1	Algorithm 2	Algorithm 3	Algorithm 4	Algorithm 5	SSM
20	**405.10**	706.50	585.60	1032.40	1106.30	1566
30	**436.00**	851.20	751.00	1230.20	2043.80	5134
40	**523.10**	1136.80	974.20	1449.60	2643.10	5949
50	**726.30**	1535.80	1217.10	1820.20	3163.60	8554
60	**905.50**	1921.80	1435.50	2472.60	3645.40	14114
70	**1466.10**	2718.70	190790	2918.50	4291.90	19290
80	**1972.20**	3365.20	2517.40	3512.40	4911.90	24602
90	3619.70	4061.80	**3184.60**	3894.90	5608.30	34801
100	4572.10	5449.30	**3756.40**	4215.40	6025.50	43130
200	68218.50	74286.80	**67614.60**	68820.80	71036.30	1686487
500	191526.10	169184.50	**161553.40**	166792.10	172968.20	11743140
1000	549218.80	399442.40	**383769.60**	395768.50	397508.40	45882769

Table 4. Number of samples taken from different test graphs using the proposed algorithms and using SSM at confidence level 0.7.

Graph size	Algorithm 1	Algorithm 2	Algorithm 3	Algorithm 4	Algorithm 5	SSM
20	**463.80**	1021.60	659.10	1237.50	1713.60	2368
30	**878.50**	1251.20	918.60	1456.70	2347.60	7763
40	1272.50	1523.10	**1252.50**	1525.40	2945.20	9004
50	1823.30	1974.00	**1556.60**	2097.80	3565.20	12956
60	3172.60	3285.80	**197470**	2837.10	4328.80	21366
70	4071.30	4669.40	**2625.70**	3772.20	5195.10	29209
80	6532.50	6751.40	**3871.10**	4946.80	5913.60	37262
90	7979.80	8168.30	**4984.20**	6259.30	6492.20	52684
100	9491.60	9613.30	**5527.70**	6658.60	6842.50	66023
200	73051.10	78085.70	**72543.10**	74214.50	76042.30	2554148
500	224362.80	188213.20	**179732.50**	178042.10	179738.90	17784157
1000	681027.50	476894.00	**462821.40**	465712.20	179738.30	69487187

- For all algorithms, the total number of samples taken from a graph is fewer than the number of samples taken using standard sampling method (SSM) for all cases of graph size and confidence level.
- From Table 7, we may observe that Algorithm 3 and Algorithm 4 require fewer number of samples than other algorithms for large graphs and higher confidence level and Algorithm 1 requires fewer samples than Algorithm 2 and Algorithm 5 for small graphs and lower confidence level.
- According to statistical significance reported by Table 7, in terms of total number of samples, Algorithm 3, Algorithm 4, Algorithm 5 Algorithm 2, and Algorithm 1 are ranked 1, 2, 3, 4 and 5, respectively.

Table 5. Number of samples taken from different test graphs using the proposed algorithms and using SSM at confidence level 0.8.

Graph size	Algorithm 1	Algorithm 2	Algorithm 3	Algorithm 4	Algorithm 5	SSM
20	**533.20**	1405.10	846.30	1328.30	1998.80	3630
30	1295.70	2481.20	**1104.20**	1758.40	2449.90	11894
40	2384.50	2936.20	**1457.50**	1972.50	3288.50	13791
50	4125.10	4519.90	**1654.70**	2217.90	3951.30	19824
60	5638.20	5749.80	**2015.90**	3015.00	4698.50	32722
70	6579.50	7102.00	**2963.00**	3982.90	5354.60	44733
80	7441.30	7719.40	**3982.10**	5177.20	6365.00	57062
90	9148.40	9463.20	**5128.30**	6285.30	6785.40	80687
100	10587.50	10394.30	**5948.80**	6858.40	7561.50	101106
200	76971.50	81473.90	**74141.20**	76901.20	79428.70	3912189
500	255981.10	216197.10	**202143.60**	211904.10	215637.40	27239090
1000	895743.30	591390.50	**559804.80**	579025.90	585243.30	106431063

Table 6. Number of samples taken from different test graphs using the proposed algorithms and using SSM at confidence level 0.9.

Graph size	Algorithm 1	Algorithm 2	Algorithm 3	Algorithm 4	Algorithm 5	SSM
20	**642.50**	2874.20	955.30	1400.50	2077.80	5976
30	1637.20	3002.50	**1309.40**	1898.50	2681.50	19585
40	2506.30	3617.00	**1656.50**	2219.70	3342.00	22704
50	4823.50	4705.60	**1817.20**	2487.80	4019.10	32652
60	6138.00	5836.70	**2492.10**	3487.20	4960.30	53868
70	7241.90	7355.40	**3526.20**	4176.30	5736.40	73652
80	8247.40	8267.10	**4270.80**	5281.70	6695.60	93960
90	9664.30	9913.70	**5765.60**	6571.40	7178.50	132843
100	11913.60	11765.50	**6329.50**	7279.10	7919.40	166491
200	82668.50	86179.60	**78379.40**	81614.10	82795.40	6442533
500	304198.20	240596.50	**223174.40**	235703.40	237175.60	448564358
1000	1108915.70	768518.30	**246319.30**	756754.00	759081.30	175268516

Table 7. The *t*-Test results of comparing algorithms in term of the number of samples taken from test graphs for different graph size

t-Test result \ Graph size	20	30	40	50	60	70	80	90	100	200	500	1000	
Algorithm 1 vs. *Algorithm* 2	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**s-**	**s-**	**s-**	**+6**
Algorithm 1 vs. *Algorithm* 3	**s+**	**s-**	**s-**	**s-**	**s-**	**s-**	**s-**	**s-**	**s-**	**s-**	**s-**	**s-**	**-10**
Algorithm 1 vs. *Algorithm* 4	**s+**	**s+**	**s-**	**s-**	**s-**	**s-**	**s-**	**s-**	**s-**	**s-**	**s-**	**s-**	**-8**
Algorithm 1 vs. *Algorithm* 5	**s+**	**s+**	**s+**	**s-**	**s-**	**s-**	**s-**	**s-**	**s-**	**s-**	**s-**	**s-**	**-6**
Algorithm 1 vs. *SSM*	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**+12**
Algorithm 2 vs. *Algorithm* 3	**s-**	**s-**	**s-**	**s-**	**s-**	**s-**	**s-**	**s-**	**s-**	**s-**	**s-**	**s-**	**-12**
Algorithm 2 vs. *Algorithm* 4	**s-**	**s-**	**s-**	**s-**	**s-**	**s-**	**s-**	**s-**	**s-**	**s-**	**s-**	**s-**	**-12**
Algorithm 2 vs. *Algorithm* 5	**s-**	**s-**	**s-**	**s-**	**s-**	**s-**	**s-**	**s-**	**s-**	**s-**	**s-**	**s-**	**-12**
Algorithm 2 vs. *SSM*	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**+12**
Algorithm 3 vs. *Algorithm* 4	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**+12**
Algorithm 3 vs. *Algorithm* 5	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**+12**
Algorithm 3 vs. *SSM*	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**+12**
Algorithm 4 vs. *Algorithm* 5	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**+12**
Algorithm 4 vs. *SSM*	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**+12**
Algorithm 5 vs. *SSM*	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**s+**	**+12**

4.2.4. *Experiment IV*

This experiment is conducted to compare Algorithm 1 with Algorithm1 in which the leaning automaton residing in each vertex is replaced by a pure chance automaton. In pure chance automaton the actions are always chosen with equal probabilities.[44] The comparison is made with respect to *DT* scaled between (0, 1) by dividing *DT* by the weight of maximum clique. The plot of *DT* versus *k* given in Fig. 8 shows the role of learning automata in guiding the process of sampling from edges of the graph for finding the stochastic maximum clique. With the aid of learning automata the process of finding the stochastic maximum clique is done with fewer numbers of samples taken from the graph as compared to the case where learning is absent. Similar results for other algorithms can also be obtained.

4.2.5. *Experiment V*

In this experiment, we study the convergence behavior of the proposed algorithms. For this purpose, we plot *PC* for the clique with maximum expected weight against iteration number *k* as given in Fig. 9. From the results one can conclude that Algorithm 3 has the highest convergence speed for all test graphs. For large graphs (RNG-50, RNG-100, RNG-200, RNG-500, and RNG-1000) Algorithm 1 has the lowest speed whereas for small graphs (Alex1-B, Alex2-B and Alex3-B) Algorithm 2 has the lowest speed of convergence. From Fig. 9, we can say that in terms of speed of convergence, Algorithm 3, Algorithm 4, Algorithm 5, Algorithm 1 and Algorithm 2, has rank 1, 2, 3, 4 and 5, respectively, for small graphs and Algorithm 3, Algorithm 4, Algorithm 5, Algorithm 2 and Algorithm 1 has rank 2, 3, 4, and 5, respectively, for large graphs.

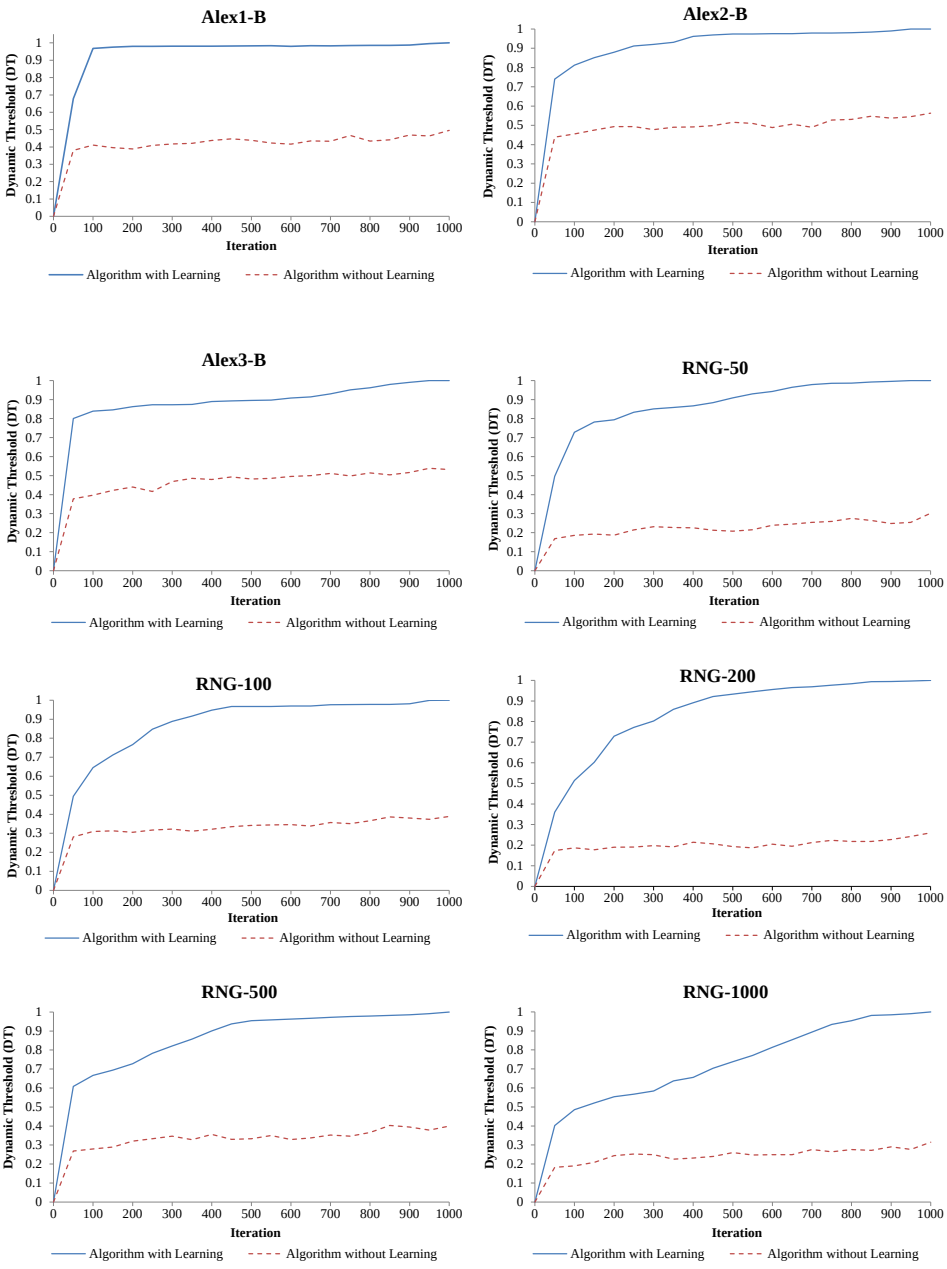


Fig. 8. Comparison of Algorithm 1 with Algorithm 1 in which learning automata are replaced with pure chance automata.

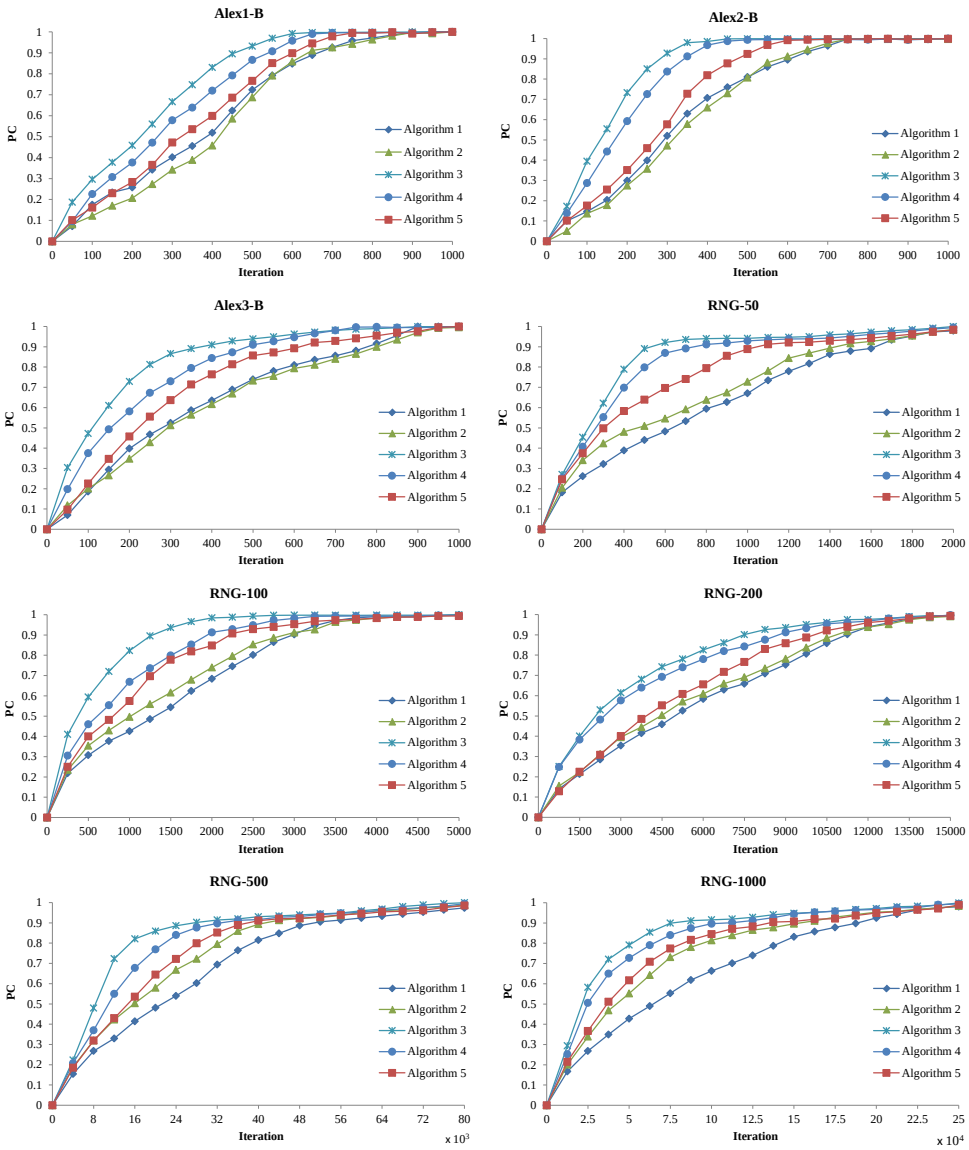


Fig. 9. Comparison of convergence behaviors of the proposed algorithms.

4.2.6. *Experiment VI*

This experiment is conducted to study the impact of learning rate (a) on the accuracy of Algorithm 1. For this purpose, we plot the percentage of runs converged to the maximum stochastic clique (PCR) for the test graphs for different learning rate (a). From the results of this experiment which is given in Fig. 10, we may conclude that for all test graphs, large value for learning rate results in higher accuracy and small value for learning rate results in lower accuracy. It is also noted from the experiments that, by a proper choice of

learning rate for Algorithm 1, we can make the probability of finding maximum stochastic clique as close to unity as possible.

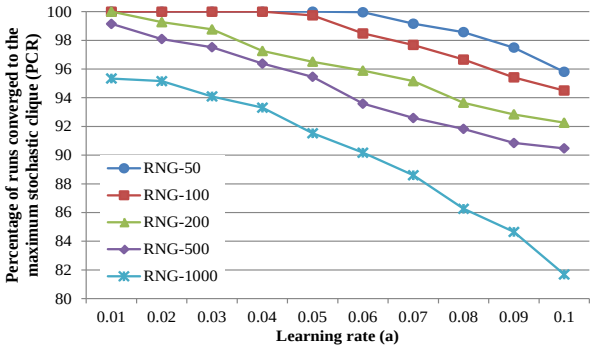
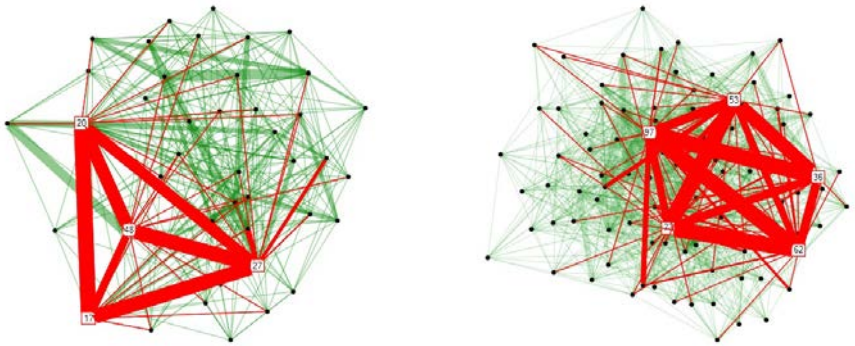
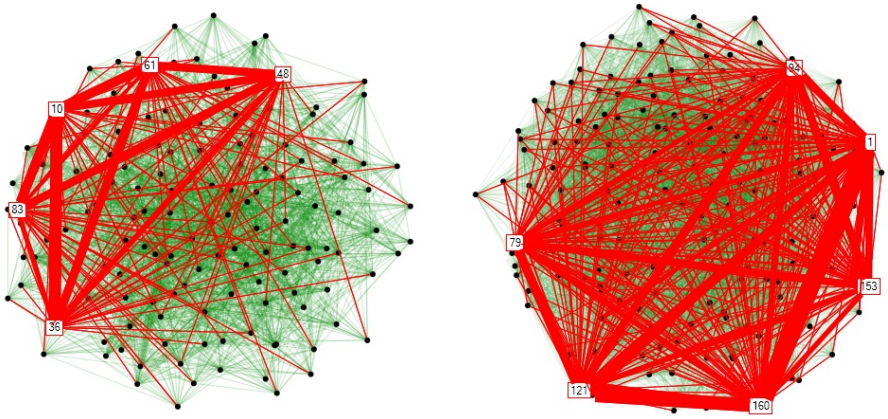


Fig. 10. Effect of varying learning rate of algorithm 1 in terms of PCR.



RNG-50 RNG-100



RNG-150 RNG-200

Fig. 11. Edges of the test graphs from which the samples are taken.

4.2.7. *Experiment VII*

Using this experiment, we try to show that the proposed algorithm gradually finds its way to the clique with maximum expected weight. A thicker edge is an edge from which more samples are taken. As shown in Fig. 11, one may observe that the edges which are along the paths to the maximum clique and the edges of the clique itself are thicker. For example, for graph RNG-50 edges between the nodes of the maximum clique ({17, 20, 27, 48}) are thicker than the other edges. Note that the edges of the maximum clique are much thicker than the other edges.

5. Conclusion

In this paper, it was argued that due to unpredictable and time varying nature of real networks, stochastic graphs is a better model for such networks and then maximum clique in stochastic graph was defined and several distributed learning automata-based algorithms were proposed for solving maximum clique problem in stochastic graph when the probability distribution functions of the weights associated with the edges of the stochastic graph are unknown. It was shown that by a proper choice of the parameters of the proposed algorithms, one can make the probability of finding maximum clique in stochastic graph as close to unity as possible. Experimental results showed that the proposed algorithms significantly reduce the number of samples needed to be taken from the edges of the stochastic graph as compared to the number of samples needed by standard sampling method at a given confidence level.

Appendix 1

In this section, we describe the estimation method of learning rate a in order to reach a given confidence level $1-\varepsilon$. Before describing the estimation method of learning rate, we need to prove the following theorem.

Theorem 1. *Let $q_i(k)$ be the probability of formation the stochastic maximum clique θ_i at iteration k, and $1-\varepsilon$ is the probability with which proposed Algorithm 1 converges to the clique θ_i,* if *$q(k)$ is updated according to proposed Algorithm 1, then there exist a learning rate $a \in (\varepsilon, q)$ for every error parameter $\varepsilon \in (0,1)$, so that*

$$\frac{xa}{e^{xa}-1} = \max_{j \neq i}\left(\frac{d_j}{d_i}\right) \tag{11}$$

where d_i is the reward probability of action α_I, $1-e^{-xq_i} = (1-e^{-x})(1-\varepsilon)$ and $q_i = [q_i(k)|k=0]$.

Proof. Let $V[u]$ is defined as

$$V[u] = \begin{cases} \dfrac{e^u - 1}{e^u} & u \neq 0 \\ 1 & u = 0 \end{cases} \tag{12}$$

and also according to Ref. 44, we have

$$\frac{1}{V[x]} = \max_{j \neq i} \left(\frac{d_j}{d_i} \right) \tag{13}$$

It has been proved in Refs. 44 and 57 that there always exists a $x > 0$ under which the Eq. (13) is satisfied, if $\frac{d_j}{d_i} < 1$, for all $j \neq i$. then we can conclude that

$$\frac{e^{-xq_i/a} - 1}{e^{-x/a} - 1} \leq \Gamma_i(q) \leq \frac{1 - e^{-xq_i}}{1 - e^{-x}} \tag{14}$$

where $\Gamma_i(q)$ is the probability with which the proposed Algorithm 1 converges to the unit vector e_i with initial vector q and defined as

$$\Gamma_i(q) = prob[q_i(\infty) = 1 | q(0) = q] = prob[q^* = e_i | q(0) = q] \tag{15}$$

q_i is the initial probability of the stochastic maximum clique θ_i. It is assumed that the probability with which the proposed Algorithm 1 with learning rate a^* converges to the stochastic maximum clique θ_i is $1-\varepsilon$, for each $0 < a < a^*$, where $a^*(\varepsilon) \in (0,1)$. So it is concluded that,

$$\frac{1 - e^{-xq_i}}{1 - e^{-x}} = 1 - \varepsilon \tag{16}$$

It is also assumed that for every error parameter $\varepsilon \in (0, 1)$ there exists a value x under which the Eq. (13) is satisfied. So, we have

$$\frac{ax}{e^{ax} - 1} = \max_{j \neq i} \left(\frac{d_j}{d_i} \right) \tag{17}$$

It is concluded that there exists a learning rate $a \in (\varepsilon, q)$ for every error parameter $\varepsilon \in (0,1)$ under which the probability with which the proposed Algorithm 1 is converged to stochastic maximum clique is greater than $1-\varepsilon$ and so Theorem 1 is proved. □

According to the above theorem, in order to reach a confidence level $1-\varepsilon$ by the proposed Algorithm 1, the learning rate a must be estimated by solving Eq. (17) based on parameter x.

In the rest of this section, the standard sampling method (SSM) is described in order to reach a given confidence level.

Theorem 2. *To obtain a confidence level not smaller than $1-\varepsilon$ for the maximum clique, it is sufficient to build a confidence with level $1-\varepsilon_i$ for every edge e_i such that $\sum_{i=1}^{n} \varepsilon_i = \varepsilon$, where n is the number of edges of clique.*

Proof. The required number of samples for each edge of graph to satisfy a confidence level $1-\varepsilon_i$ is obtained by using edge sampling method described below.

Edge sampling method. Let $(x_1, x_2, \ldots, x_N)$ be a random sample of random variable X_i associated with weight of edge e_i with unknown mean μ and variance σ^2. If $\bar{x} \pm \sigma / \sqrt{N \varepsilon_i}$, where

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{18}$$

is a $(1-\varepsilon_i)$% confidence interval for mean μ, then for each sufficiently small value of δ, there exist a positive number N_0 such that

$$p[|\bar{x}_N - \mu| < \delta] \geq 1 - \varepsilon \tag{19}$$

for all $N \geq N_0$.

The problem is then to find a confidence region for each of edges under which a desired confidence level $1-\varepsilon$ is guaranteed for the maximum clique. The confidence region for the maximum clique is defined as the intersection $\bigcap_{i=1}^{n} C_i(\varepsilon_i)$ of the confidence regions for the edges, where $C_i(\varepsilon_i) = 1 - \varepsilon_i$ is the confidence region for e_i and n is the number of edges of maximum clique. Using *Booles-Bonferroni* inequality,[58] we have

$$\min_{1\leq i\leq n}(1-\varepsilon_i) > p\left(\mu_i \epsilon \bigcap_{i=1}^{n} C_i(\varepsilon_i)\right) \geq 1 - \sum_{i=1}^{n} p[\mu_i \notin C_i(\varepsilon_i)] \tag{20}$$

and so

$$\min_{1\leq i\leq n}(1-\varepsilon_i) > p\left(\mu_i \epsilon \prod_{i=1}^{n} C_i(\varepsilon_i)\right) \geq 1 - \sum_{i=1}^{n} \varepsilon_i \tag{21}$$

Hence, the confidence level of the maximum clique is not smaller than $1 - \sum_{i=1}^{n} \varepsilon_i$. In this theorem, the goal is to obtain a confidence level not smaller than $1-\varepsilon$ for the maximum clique. To achieve this, according to the Bonferroni Correction[59] it is sufficient to build a confidence with level $1-\varepsilon_i$ for each edge e_i such that $\sum_{i=1}^{n} \varepsilon_i = \varepsilon$, and hence the proof of the Theorem 2. □

Appendix 2

Probability distribution function of small graphs (Alex1-B, Alex2-B, Alex3-B)[55] are listed as following tables.

Table 8. Stochastic graph *Alex1-B.*

Edge	Weight	Probabilities
(v_1, v_2)	{2, 8, 12}	{0.9, 0.08, 0.02}
(v_1, v_3)	{10, 24, 35}	{0.85, 0.12, 0.03}
(v_1, v_4)	{6, 18, 21}	{0.88, 0.1, 0.02}
(v_2, v_3)	{12, 22, 30}	{0.85, 0.11, 0.04}
(v_2, v_5)	{17, 35, 50}	{0.75, 0.2, 0.05}
(v_3, v_4)	{3, 7, 12}	{0.84, 0.13, 0.03}
(v_3, v_6)	{10, 19, 24}	{0.8, 0.14, 0.06}
(v_3, v_7)	{4, 6, 10}	{0.75, 0.17.0.08}
(v_4, v_7)	{9, 13, 20}	{0.82, 0.15, 0.03}
(v_4, v_8)	{21, 33, 45}	{0.85, 0.09, 0.06}
(v_5, v_6)	{8, 14, 18}	{0.77, 0.13, 0.10}
(v_6, v_7)	{20, 30, 40}	{0.87, 0.09, 0.04}
(v_7, v_8)	{10, 17, 30}	{0.95, 0.03, 0.02}
(v_6, v_8)	{15, 28, 45}	{0.79, 0.15, 0.06}

Table 9. Stochastic graph *Alex2-B.*

Edge	Weight	Probabilities
(v_1, v_2)	{2, 8, 12}	{0.9, 0.08, 0.02}
(v_1, v_3)	{6, 18, 21}	{0.88, 0.1, 0.02}
(v_2, v_4)	{17, 35, 50}	{0.75, 0.2, 0.05}
(v_2, v_5)	{3, 7, 10}	{0.68, 0.25, 0.07}
(v_3, v_7)	{9, 13, 20}	{0.82, 0.15, 0.03}
(v_3, v_8)	{21, 33, 45}	{0.85, 0.09, 0.06}
(v_4, v_5)	{18, 27, 36}	{0.94, 0.05, 0.01}
(v_4, v_6)	{8, 14, 18}	{0.77, 0.13, 0.10}
(v_4, v_9)	{15, 21, 25}	{0.76, 0.22, 0.02}
(v_5, v_6)	{25, 38, 50}	{0.8, 0.12, 0.08}
(v_6, v_7)	{20, 30, 40}	{0.87, 0.09, 0.04}
(v_6, v_8)	{10, 17, 30}	{0.95, 0.03, 0.02}
(v_6, v_9)	{4, 19, 15}	{0.75, 0.14, 0.11}
(v_7, v_8)	{15, 28, 45}	{0.79, 0.15, 0.06}
(v_8, v_9)	{8, 14, 25}	{0.85, 0.1, 0.05}

Table 10. Stochastic Graph *Alex3-B.*

Edge	Weight	Probabilities
(v_1, v_2)	{2, 8, 12}	{0.9, 0.08, 0.02}
(v_1, v_3)	{10, 24, 35}	{0.85, 0.12, 0.03}
(v_1, v_4)	{6, 18, 21}	{0.88, 0.1, 0.02}
(v_2, v_5)	{17, 35, 50}	{0.75, 0.2, 0.05}
(v_2, v_6)	{3, 7, 12}	{0.68, 0.25, 0.07}
(v_2, v_3)	{12, 22, 30}	{0.85, 0.11, 0.04}
(v_3, v_7)	{10, 19, 24}	{0.8, 0.14, 0.06}
(v_3, v_8)	{4, 6, 10}	{0.75, 0.17.0.08}
(v_3, v_4)	{3, 7, 12}	{0.84, 0.13, 0.08}
(v_4, v_9)	{21, 33, 45}	{0.85, 0.09, 0.06}
(v_5, v_7)	{8, 14, 18}	{0.77, 0.13, 0.10}
(v_5, v_{10})	{15, 21, 25}	{0.76, 0.22, 0.02}
(v_3, v_6)	{5, 10, 12}	{0.65, 0.23, 0.12}
(v_5, v_6)	{18, 27, 36}	{0.94, 0.05, 0.01}
(v_6, v_7)	{25, 38, 50}	{0.8, 0.12, 0.08}
(v_7, v_8)	{20, 30, 40}	{0.87, 0.09, 0.04}
(v_7, v_{10})	{4, 19, 15}	{0.75, 0.14, 0.11}
(v_4, v_8)	{9, 13, 20}	{0.82, 0.15, 0.03}
(v_8, v_9)	{15, 28, 45}	{0.79, 0.15, 0.06}
(v_7, v_9)	{10, 17, 30}	{0.95, 0.03, 0.02}
(v_9, v_{10})	{8, 14, 25}	{0.85, 0.1, 0.05}

References

1. P. Erdos and A. Rényi, On the evolution of random graphs, *Publications of the Mathematical Institute of the Hungarian Academy of Sciences* **5** (1960) 17–61.
2. D. J. Watts and S. H. Strogatz, Collective dynamics of "small-world" networks, *Nature* **393** (1998) 440–442.
3. A. L. Barabási and R. Albert, Emergence of scaling in random networks, *Science* **286** (1999) 509–512.
4. Y. Zhang, Z.-P. Fan and Y. Liu, A method based on stochastic dominance degrees for stochastic multiple criteria decision making, *Computers & Industrial Engineering* **58** (2010) 544–552.
5. P. Vasant, I. Elamvazuthi and J. F. Webb, Fuzzy technique for optimization of objective function with uncertain resource variables and technological coefficients, *International Journal of Modeling, Simulation, and Scientific Computing* **1** (2010) 349–367.
6. P. Vasant, Solving fuzzy optimization problems of uncertain technological coefficients with genetic algorithms and hybrid genetic algorithms pattern search approaches, *Innovation in Power, Control, and Optimization: Emerging Energy Technologies* **396** (2012) 344–368.
7. I. Elamvazuthi, P. Vasant and T. Ganesan, Hybrid Optimization Techniques for Optimization in a Fuzzy Environment, in *Handbook of Optimization* (Springer, 2013), pp. 1025–1046.
8. T. Ganesan, P. Vasant and I. Elamvazuthi, Hopfield neural networks approach for design optimization of hybrid power systems with multiple renewable energy sources in a fuzzy environment, *Journal of Intelligent and Fuzzy Systems* **26** (2014) 2143–2154.
9. Y.-K. Lin and C.-F. Huang, Backup reliability of stochastic imperfect-node computer networks subject to packet accuracy rate and time constraints, *International Journal of Computer Mathematics* **90** (2013) 457–474.
10. H. J. Hwang and J. J. Velázquez, Bistable stochastic biochemical networks: large chemical networks and systems with many molecules, *Journal of Mathematical Chemistry* **51** (2013) 2074–2103.
11. Y. Ni and Q. Shi, Minimizing The Complete Influence Time in a Social Network With Stochastic Costs For Influencing Nodes, *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* **21** (2013) 63–74.
12. C. Liu and Z.-K. Zhang, Information spreading on dynamic social networks, *Communications in Nonlinear Science and Numerical Simulation* **19** (2014) 896–904.
13. A. Rezvanian and M. R. Meybodi, Sampling social networks using shortest paths, *Physica A: Statistical Mechanics and Its Applications* **in press** (2015) 1–15.
14. L. Jin, Y. Chen, T. Wang, P. Hui and A. V. Vasilakos, Understanding user behavior in online social networks: A survey, *IEEE Communications Magazine* **51** (2013) 144–150.
15. M. S. Granovetter, The strength of weak ties, *American Journal of Sociology* (1973) 1360–1380.
16. H. Beigy and M. R. Meybodi, Utilizing distributed learning automata to solve stochastic shortest path problems, *International Journal of Uncertainty Fuzziness And Knowledge Based Systems* **14** (2006) 591–615.
17. L. Yang, X. Yang and C. You, Stochastic scenario-based time-stage optimization model for the least expected time shortest path problem, *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* **21** (2013) 17–33.
18. J. Akbari Torkestani and M. R. Meybodi, Learning automata-based algorithms for finding minimum weakly connected dominating set in stochastic graphs, *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* **18** (2010) 721–758.
19. R. M. Karp, Reducibility among combinatorial problems, *Complexity of Computer Computations* **40** (1972) 85–103.
20. Q. Wu and J.-K. Hao, A review on algorithms for maximum clique problems, *European Journal of Operational Research* **242** (2015) 693–709.

21. T. S. Motzkin and E. G. Straus, Maxima for graphs and a new proof of a theorem of Turán, *Canadian Journal of Mathematics* **17** (1965) 533–540.
22. A. Massaro, M. Pelillo and I. M. Bomze, A complementary pivoting approach to the maximum weight clique problem, *SIAM Journal on Optimization* **12** (2002) 928–948.
23. J. Cheng, Y. Ke, A. W. C. Fu, J. X. Yu and L. Zhu, Finding maximal cliques in massive networks, *ACM Transactions on Database Systems (TODS)* **36** (2011) 21.
24. L. Babel and G. Tinhofer, A branch and bound algorithm for the maximum clique problem, *Operations Research* **34** (1990) 207–217.
25. R. Carmo and A. Züge, Branch and bound algorithms for the maximum clique problem under a unified framework, *Journal of the Brazilian Computer Society* **18** (2012) 137–151.
26. E. Balas and W. Niehaus, Optimized crossover-based genetic algorithms for the maximum cardinality and maximum weight clique problems, *Journal of Heuristics* **4** (1998) 107–122.
27. M. Brunato and R. Battiti, R-EVO: A Reactive Evolutionary Algorithm for the Maximum Clique Problem, *IEEE Transactions on Evolutionary Computation* **15** (2011) 770 –782.
28. C. Solnon and S. Fenet, A study of ACO capabilities for solving the maximum clique problem, *Journal of Heuristics* **12** (2006) 155–180.
29. X. Geng, J. Xu, J. Xiao and L. Pan, A simple simulated annealing algorithm for the maximum clique problem, *Information Sciences* **177** (2007) 5064–5071.
30. Q. Wu and J.-K. Hao, An adaptive multistart tabu search approach to solve the maximum clique problem, *Journal of Combinatorial Optimization* **26** (2013) 86–108.
31. B.-D. Zhou, H.-L. Yao, M.-H. Shi, Q. Yue and H. Wang, An new immune genetic algorithm based on uniform design sampling, *Knowledge and Information Systems* **31** (2012) 389–403.
32. R. Battiti and M. Protasi, Reactive local search for the maximum clique problem, *Algorithmica* **29** (2001) 610–637.
33. K. Katayama, A. Hamamoto and H. Narihisa, An effective local search for the maximum clique problem, *Information Processing Letters* **95** (2005) 503–511.
34. U. Benlic and J.-K. Hao, Breakout local search for maximum clique problems, *Computers & Operations Research* **40** (2013) 192–206.
35. M. Soleimani-pouri, A. Rezvanian and M. R. Meybodi, An Ant Based Particle Swarm Optimization Algorithm for Maximum Clique Problem in Social Networks, in *State of the Art Applications of Social Network Analysis*, eds. F. Can, T. Özyer and F. Polat (Springer International Publishing, 2014), pp. 295–304.
36. M. Soleimani-Pouri, A. Rezvanian and M. R. Meybodi, Finding a Maximum Clique Using Ant Colony Optimization and Particle Swarm Optimization in Social Networks, in *Proceedings of the 2012 International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2012)* (IEEE Computer Society, 2012), pp. 58–61.
37. W. Pullan, F. Mascia and M. Brunato, Cooperating local search for the maximum clique problem, *Journal of Heuristics* **17** (2011) 181–199.
38. M. Bhattacharyya and S. Bandyopadhyay, Solving maximum fuzzy clique problem with neural networks and its applications, *Memetic Computing* **1** (2009) 281–290.
39. J. Liu and Y. T. Lee, Graph-based method for face identification from a single 2D line drawing, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **23** (2001) 1106–1119.
40. Q. Yang, P. Zhou, H. Zhang and J. Zhang, Clique Discovery Based on User Similarity for Online Shopping Recommendation, *Information Technology Journal* **10** (2011) 1587–1593.
41. S. Mimaroglu and M. Yagci, CLICOM: Cliques for combining multiple clusterings, *Expert Systems With Applications* **39** (2011) 1889–1901.
42. Y. Liu, D. Zhang and F. Y. L. Chin, A clique-based algorithm for constructing feasible timetables, *Optimization Methods & Software* **26** (2011) 281–294.
43. L. Wang, R. Wei, Y. Lin and B. Wang, A clique base node scheduling method for wireless sensor networks, *Journal of Network and Computer Applications* **33** (2010) 383–396.
44. K. S. Narendra and M. A. L. Thathachar, *Learning Automata: An Introduction* (Printice-Hall, 1989).

45. A. Mousavian, A. Rezvanian and M. R. Meybodi, Solving Minimum Vertex Cover Problem Using Learning Automata, in *Proceedings of 13th Iranian Conference on Fuzzy Systems (IFSC 2013)* (2013), pp. 1–5.
46. A. Mousavian, A. Rezvanian and M. R. Meybodi, Cellular learning automata based algorithm for solving minimum vertex cover problem, in *2014 22nd Iranian Conference on Electrical Engineering (ICEE)* (IEEE, 2014), pp. 996–1000.
47. D. Maravall and J. de Lope, Fusion of learning automata theory and granular inference systems: ANLAGIS. Applications to pattern recognition and machine learning, *Neurocomputing* **74** (2011) 1237–1242.
48. M. Hasanzadeh, M. R. Meybodi and M. M. Ebadzadeh, Adaptive cooperative particle swarm optimizer, *Applied Intelligence* **39** (2013) 397–420.
49. H. Morshedlou and M. R. Meybodi, Decreasing Impact of SLA Violations: A Proactive Resource Allocation Approach for Cloud Computing Environments, *IEEE Transactions on Cloud Computing* **2** (2014) 156–167.
50. H. Beigy and M. R. Meybodi, Adaptive limited fractional guard channel algorithms: a learning automata approach, *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* **17** (2009) 881–913.
51. A. Rezvanian, M. Rahmati and M. R. Meybodi, Sampling from complex networks using distributed learning automata, *Physica A: Statistical Mechanics and Its Applications* **396** (2014) 224–234.
52. M. Thathachar, Learning automata with changing number of actions, *IEEE Transactions on Systems, Man, and Cybernetics* **17** (1987) 1095–1100.
53. J. Akbari Torkestani and M. R. Meybodi, Clustering the wireless Ad Hoc networks: A distributed learning automata approach, *Journal of Parallel and Distributed Computing* **70** (2010) 394–405.
54. M. Hasanzadeh and M. R. Meybodi, Grid resource discovery based on distributed learning automata, *Computing* **96** (2014) 909–922.
55. K. R. Hutson and D. R. Shier, Minimum spanning trees in networks with varying edge weights, *Annals of Operations Research* **146** (2006) 3–18.
56. J. Akbari Torkestani and M. R. Meybodi, Finding minimum weight connected dominating set in stochastic graph based on learning automata, *Information Sciences* **200** (2012) 57–77.
57. S. Lakshmivarahan and M. A. L. Thathachar, Bounds on the convergence probabilities of learning automata, *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* **6** (1976) 756–763.
58. F. B Alt, Bonferroni inequalities and intervals, *Encyclopedia of Statistical Sciences* (1982).
59. C. E. Bonferroni, *Teoria statistica delle classi e calcolo delle probabilita* (Libreria internazionale Seeber, 1936).