

# Finding Minimum Vertex Covering in Stochastic Graphs: A Learning Automata Approach

Alireza Rezvanian, Mohammad Reza Meybodi

*Soft computing laboratory, Department of Computer Engineering and Information Technology  
Amirkabir University of Technology (Tehran Polytechnic), Tehran, Iran*

**Abstract:** Structural and behavioral parameters of many real networks such as social networks are unpredictable, uncertain and time varying parameters and for these reasons deterministic graphs for modeling such networks are too restrictive to solve most of the real network problems. It seems that stochastic graphs, in which weights associated to the vertices are random variable, may be a better graph model for real world networks. Once we use stochastic graph as the model for a network, every feature of the graph such as path, spanning tree, clique, dominating set and cover set should be treated as a stochastic feature. For example, choosing stochastic graph as a graph model of an online social network and defining community structure in terms of clique, the concept of stochastic clique may be used to study community structures properties or defining influence spreading according to the coverage of influential users, the concept of stochastic vertex covering may be used to study spread of influence. In this paper, minimum vertex covering in stochastic graphs is first defined and then four learning automata-based algorithms for solving minimum vertex covering problem in stochastic graphs where the probability distribution functions of the weights associated with the vertices of the graph are unknown are proposed. It is shown that by a proper choice of the parameters of the proposed algorithms, one can make the probability of finding minimum vertex cover in stochastic graph as close to unity as possible. Experimental results on synthetic stochastic graphs reveal that at a certain confidence level the proposed algorithms significantly outperform standard sampling method in terms of the number of samples needed to be taken from the vertices of the stochastic graph.

**Keywords:** Covering problem, Minimum vertex covering, Stochastic graph, Learning automata, Social networks.

## 1. INTRODUCTION

To study real-world networks such as computer networks, biochemical reaction networks, transportation networks, and social networks several graph models such as Erdős-Rényi model (Erdos and Rényi 1960), Watts-Strogatz model (Watts and Strogatz 1998) and Barabási-Albert model (Barabási and Albert 1999) are introduced in the literature. These graph models are deterministic and too restrictive to solve most of the real network problems due to their unpredictable, uncertain and time varying nature. Considering a deterministic graph with fixed weight associated with the vertices as a model of real world networks is not suitable when tasks/activities on networks vary with time. For example in online social networks, the activities of users such as friendship behavior, liking a comment, retweeting a tweet in Twitter and frequency of taking a comment on a wall post in Facebook vary over time with unknown probabilities (Jin et al. 2013). Analyzing social networks with deterministic graph models 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 or vertex weights are assumed to be fixed considers only a snapshot of a real-world network. Moreover, in analyzing online social networks not only understanding the structure and topology of the network is important but also the degree of association among the users in network is significant for several analytical applications of social networks.

In recent years, the emergence of online social networks has been attracted widespread attention and many problems with various applications have been appeared for researchers (Yang et al. 2012; Liu, Hsu, and Ke 2013; Liaqat et al. 2014; Li et al. 2014; Najaflou et al. 2014). Maximization of the spread of influence in social network is the problem of finding a set of initial influential spreaders that maximize the spread of information through social network under a given cascading model motivated by many companies to

introduce and promote their new products, services, innovations and technologies (Kempe, Kleinberg, and Tardos 2003). Analyzing the characteristics of social relationship among users plays an important role to attain such objectives on online social networks. The strength of influential spreaders as influential leaders of network which may not have necessarily the maximum number of friends, because of its higher weight impresses on the weaker users to accept, adopt and spread certain information. Deterministic view about the social strength of users and their activities only focuses on the impact of spreaders and also ignore the time varying nature of such users and activities.

Online social networks (OSN) providers such as *Facebook* and *Twitter* need to measure several parameters of their networks for analytical goals via monitoring the activities of users and their relationships. Also, by monitoring activities and participation of users of OSNs, common and different usage patterns can be discovered for marketing goals (Gyarmati and Trinh 2010). Although the direct monitoring the status of all users due to large size of network cannot be feasible in a limited amount of time and is too expensive, this process can be done by vertex cover of users with minimum cost (Yu 2012). In monitoring users in social networks, OSN providers need to consider the degree of importance for activities of users to reach a proper measuring the user participations for analytical approaches and also find a proper set of initial influential users for marketing approaches. The strength of user participations and their activities in network which may not consist necessarily the maximum number or value of a parameter, because of its higher weight impresses on the weaker users to participate with together or propagate a kind of activities for marketing goals. Deterministic view about the social strength of users and their activities only focuses on the impact of users and also ignore the time varying nature of such users, participations, activities and their relations.

According to the points discussed in previous paragraphs, it seems stochastic graphs in which weights associated to the vertices are random variables is a better candidate as a graph model for real-world network applications with time varying nature. By choosing stochastic graph as a graph model, every feature of the graph such as path (Misra and Oommen 2009), cover, clique, spanning tree (J. Akbari Torkestani and Meybodi 2012), dominating set (J. A. Torkestani and Meybodi 2012) and vertex cover to name a few, should be treated as stochastic features. For example, once the graph model of an online social network is chosen to be stochastic graph influence spreading using covering concept, and the influence associations among the users as random variables, the concept of stochastic cover may be more appropriate for investigating spreading properties.

In this paper, minimum vertex cover in stochastic graph is first defined and then four learning automata-based algorithms for solving it under the situation that the probability distribution functions of the weights associated with the vertices of graph are unknown are proposed. The proposed algorithms by taking samples from the vertices of the stochastic graph try to find the vertex cover which has minimum expected weight. The process of sampling from the vertices of the graph is guided in such a way that the number of samples needed to be taken from the vertices of the stochastic graph for finding the vertex cover with minimum expected weight to be reduced as much as possible. In the proposed algorithms, each vertex of the graph is equipped with a leaning automaton whose actions correspond to choosing one of the neighboring vertices of the corresponding vertex. The guided sampling process implemented by learning automata aims to take more samples from the promising vertices that is the vertices which are along the paths toward the vertex cover with minimum expected weight or the vertex cover itself instead of walking around the graph and taking unnecessary samples from non-promising vertices.

In order to study the performance of the proposed algorithms, we conducted several experimental studies on different stochastic graphs. Experimental results show that the proposed algorithms significantly outperform the standard sampling method in terms of number of samples needed to be taken from the vertices of stochastic graph in order to find the vertex cover with minimum expected weight. It is also shown that by a proper choice of the parameters of the proposed algorithms, the probability of finding the minimum vertex cover problem is as close to unity as possible. The rest of this paper is organized as follows. Section 2 dedicated to some preliminaries such stochastic graphs, learning automata, some of their variants for the research works and standard sampling method presented in this paper. The proposed algorithms for finding minimum vertex cover in stochastic graph are described in section 3 and section 4 presents the simulation results. Section 5 concludes the paper.

## 2. PRELIMINARIES

In this section, in order to provide basic information for the remainder of the paper, we present a brief overview of vertex cover problem, stochastic graph, stochastic vertex cover, learning automata, variable action set learning automata, distributed learning automata, and standard sampling method.

### 2.1. Vertex Covering

Let  $G\langle V, E \rangle$  be an undirected graph, where  $V$  denotes the set of vertices, and  $E \subseteq V \times V$  is the edge-set. A vertex cover  $V'$  is a subset of vertices that for every  $e(v_i, v_j) \in E$  at least one of the vertices  $v_i$  or  $v_j$  is an element of  $V'$  in the graph  $G$ . Minimum vertex cover (MVC) of graph  $G$  is a vertex cover with minimum size among all possible vertex cover of graph  $G$ . Minimum weight vertex cover (MWVC) problem is a fundamental graph problem defined for weighted graph. MWVC problem is the problem of seeking a vertex cover with the minimum total weight. MVC problem and MWVC problem are both known to be NP-hard (Karp 1972) and due to its natural computational complexity, several algorithms have been reported in the literature for solving it which most of them are based on heuristics for reaching near optimal solutions in a reasonable computation time. The MVC and MWVC problems are used in a wide-spread of many real-world applications such as scheduling (Kuhn and Mastrolilli 2013), facility location (Gupta et al. 2011), network flow (Bentz et al. 2012), network security (Jadliwala, Bilegrevic, and Hubaux 2013), cascading failures (Veremyev et al. 2014), belief propagation (Jin-Hua and Hai-Jun 2014), propagation of ideas through a social network (Kim et al. 2014) to name a few. For example MVC or MWVC of users in online social networks indicates a high influential set of users.

Various studies are presented in the literatures for seeking solutions of minimum vertex cover in deterministic graph (Cai et al. 2013; Bouamama, Blum, and Boukerram 2012; Mousavian, Rezvani, and Meybodi 2013) which can be classified into two groups of exact algorithms and heuristic algorithms. The first group which is based on branch and bound technique tries to find all solutions, but in large cases or limited time may fail to reach optimal solutions. The second group of algorithms which are based on heuristic techniques can reach to near optimal solution in reasonable amount of time. A famous greedy heuristic has been presented by Chvatal to collecting vertices with the smallest ratio between its weight and degree (Chvatal 1979). A generalized heuristic approach by combining genetic algorithm and greedy heuristic has been investigated by Singh et al. (Singh and Gupta 2006). Shyu et al. in (Jovanovic and Tuba 2011) proposed an ant colony based algorithm (ACO) to solve the MWVC problem. They showed through their simulations that ACO outperforms tabu search and simulated annealing. Jovanovic et al. described a parallel version of ACO (Jovanovic, Tuba, and Simian 2010). In this method, each colony assigned to a separate process and then best solutions are exchanged among search threads. Another improved version of ACO by using pheromone correction strategy in order to avoid stagnation of the search and local optima convergence are presented by Jovanovic (Jovanovic and Tuba 2011). A hybridized tabu search by incorporating reactive tabu search with simulated annealing and random walk was introduced by Voß et al. (Voß and Fink 2012) but in this study the results are very much dependent on the parameter setting. A population based iterated greedy search approach for finding vertex cover problem suggested by Bouamama et al. in (Bouamama, Blum, and Boukerram 2012). In this approach, at each iteration, a population of solutions is created and refined using a fast randomized iterated greedy heuristic method. Their simulation results revealed that a population of solutions rather than a single solution may avoid the fast local optima convergence. Another direction of studies includes local search algorithms as a greedy heuristics. In (Cai, Su, and Sattar 2011) Cai et al. introduced two local search algorithms as edge weighting local search (EWLS) and edge weighting configuration checking (EWCC). In EWLS algorithm, iteratively a partial vertex cover is updated by an edge weighting schema. However, this algorithm may be fallen in a cycle, so in EWCC the problem of cycle formation is handled. They also enhanced the edge weighting schema with inserting a new forgetting operator and several combined strategies in (Cai et al. 2013). Mousavian et al. (Mousavian, Rezvani, and Meybodi 2013) proposed a learning automaton based algorithm for solving covering problem in deterministic graph. In this algorithm, a set of learning automata cooperating with each other to select promising vertices by updating their action probabilities. Vertex cover problem has also been solved using several approximation algorithms (Boria, Della Croce, and Paschos 2014; Zhang et al. 2014).

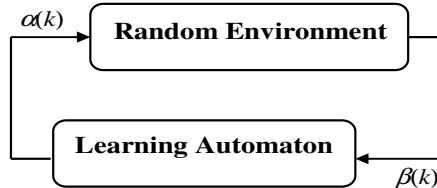
## 2.2. Stochastic graph and stochastic vertex covering

An vertex-weighted undirected graph can be described by a triple  $G(V, E, W)$  where  $V=\{v_1, v_2, \dots, v_n\}$  is the set of vertices,  $E=\{e_1, e_2, \dots, e_m\} \subseteq V \times V$  is the edge-set, and  $W=\{w_1, w_2, \dots, w_n\}$  is the set of weights associated with the vertices of graph. Graph  $G$  is said to be stochastic if weight  $w_i$  associated with vertex  $v_i$  of graph is a random variable. *Stochastic minimum weight vertex cover* in a stochastic graph is a vertex cover with minimum expected weight.

## 2.3. Learning automata

A learning automaton (LA) (Narendra and Thathachar 1989) refers to an abstract model which randomly selects one action out of its finite set of allowed 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 chosen action and responds to the automata with a reinforcement signal. Based on chosen action, and received signal, the automata updates its internal state and chooses 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  $\alpha=\{\alpha_1, \alpha_2, \dots, \alpha_r\}$  is the set of inputs (action sets),  $\beta=\{\beta_1, \beta_2, \dots, \beta_m\}$  is the set of output values that can be taken by the reinforcement signal,  $c=\{c_1, c_2, \dots, c_m\}$  is the set of penalty probabilities measured by the reaction of the environment, and  $c_i$  is the penalty probability for action  $\alpha_i$ . According to the nature of the reinforcement signal  $\beta$ , the environment can be categorized into *P-model*, *Q-model* and *S-model*. P-model refers to an environment where 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.



**Figure 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 (Narendra and Thathachar 1989). 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),  $\alpha=\{\alpha_1, \alpha_2, \dots, \alpha_r\}$  is the action sets,  $\beta=\{\beta_1, \beta_2, \dots, \beta_r\}$  denotes the set of inputs and  $p=\{p_1, p_2, \dots, p_r\}$  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 increases 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 equation (1), on the other hand, if the chosen action  $\alpha_i(k)$  is rewarded by the random environment, and it is updated according to equation (2) if the taken action is penalized.

If  $a=b$ , the recurrence equations (1) and (2) are called linear reward-penalty ( $L_{R-P}$ ) algorithm; if  $a>>b$ , they are called linear reward- $\epsilon$ -penalty ( $L_{R-\epsilon P}$ ) 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 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 (J. A. Torkestani 2012a; Javad Akbari Torkestani 2013), pattern recognition (Maravall and de Lope 2011), image processing (Cuevas, Zaldivar, and Pérez-Cisneros 2011), optimization (Moradabadi and Beigy 2014), grid systems (Javad Akbari Torkestani 2013), information retrieval (J. A. Torkestani 2012b), cloud computing (Moshedlou and Meybodi 2014), wireless sensor networks (Safavi, Meybodi, and Esnaashari 2014; Nicopolitidis 2015), ad hoc networks (Shirali, Shirali, and Meybodi 2012), and complex networks (Rezvanian, Rahmati, and Meybodi 2014).

## 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 (Thathachar 1987). Such an automaton includes finite set of  $n$  actions,  $\alpha=\{\alpha_1, \alpha_2, \dots, \alpha_n\}$ .  $A=\{A_1, A_2, \dots, A_m\}$  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  $q(k)=\{q_1(k), q_2(k), \dots, q_m(k)\}$ , an external agency chooses randomly the particular action subsets, where:  $q_i(k)=prob[A(k)=A_i|A_i \in A, 1 \leq i \leq 2^n-1]$ .  $\hat{p}_i(k)=prob[\alpha(k)=\alpha_i|A(k), \alpha_i \in A(k)]$  is the probability of choosing action  $\alpha_i$ , 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 equation (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  $\alpha_i \in A(k)$ .

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

## 2.5. Distributed Learning Automata

A Distributed learning automata (DLA) (Beigy and Meybodi 2006) 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  $A=\{A_1, A_2, \dots, A_n\}$  is the set of learning automata,  $E \subset A \times A$  is the set of edges where edge  $e_{i,j}$  corresponds to action  $\alpha_i^j$  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.

The operation of a DLA can be described as follows: At first, the root automaton  $A_0$  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 selects an action which results in activation of another automaton. The process of choosing the actions and activating the automata is continued until some stopping criteria depending on the problem for which DLA is designed are met. DLA may stop if either the action of the activated automaton is empty or all the vertices of the graph have been visited. The chosen actions along the traversed path or in the part of solutions by the activated learning automata are performed on the random environment. The environment evaluates the performed actions and emits a reinforcement signal to the DLA. The action probability vectors of the activated learning automata along the traversed path or learning automata of the nodes which are part of solution to the problem are then updated on the basis of the reinforcement signal. Activation of DLA starting from its root is repeated specified number of times until the solution of the problem for which DLA is designed is obtained. For example in a vertex with 2 adjacent vertices the automaton has two actions. If automaton  $A_i$  selects  $\alpha_i^k$  from its action set, then it will activate automaton  $A_k$ . Afterward, automaton  $A_k$  chooses one of its possible actions and so on. DLA has found successful applications in several areas including: wireless ad hoc networks (J. A. Torkestani and Meybodi 2010a), web mining (Forsati and Meybodi 2010), grid computing (Hasanzadeh and Meybodi 2014), complex networks (Rezvanian, Rahmati, and Meybodi 2014) and social network analysis (Soleimani-Pouri, Rezvanian, and Meybodi 2012).

## 2.6. Standard Sampling Method

The required sample size for a random variable to satisfy a confidence level  $1-\epsilon_i$  can be obtained using the following theorem (Papoulis and Pillai 2002).

**Theorem:** Let  $(x_1, x_2, \dots, x_N)$  be  $N$  random samples of random variable  $X_i$  with unknown mean  $\mu$  and variance  $\sigma^2$ . If  $\bar{x} \pm \sigma / \sqrt{N\epsilon_i}$ , where  $\bar{x} = \frac{1}{N} \sum_{i=1}^N x_i$  is a  $(1-\epsilon_i)\%$  confidence interval for mean  $\mu$ , then for each sufficiently small value of error  $\delta$ , there exist a positive number  $N_0$  such that  $p[|\bar{x}_N - \mu| < \delta] > 1 - \epsilon$  for all  $N \geq N_0$ .

Supplementary and additional mathematic details about standard sampling method will be provided in appendix I.

## 3. Proposed Algorithm

Let  $G(V, E, W)$  be the input stochastic graph, where  $V=\{v_1, v_2, \dots, v_n\}$  is the set of vertices,  $E$  is the edge set, and  $W=\{w_1, w_2, \dots, w_n\}$  is a set of random variables each of which is associated to a vertex of the input graph.  $n$  and  $m$  are the total number of vertices and edges in the input graph respectively. It is supposed that the weight of each vertex is a positive random variable with unknown probability distribution function. The proposed algorithm uses a network of learning automata isomorphic to the input graph and tries to find a vertex cover with minimum expected weight by sampling the vertices of the graph in such a way that the number of samples taken from the vertices of graph be minimized. An action of automaton  $A_i$  assigned to vertex  $v_i$  corresponds to the selection of one of the neighboring vertices of vertex  $v_i$ . The process of sampling from the vertices of graph is guided by the set of learning automata assigned to the vertices of the input

graph. In the proposed algorithm at any time each learning automaton can be in either active or inactive modes. The proposed algorithm consists of four steps as described below.

### 3.1. Initialization

A network of learning automata defined by 2-tuple  $\langle A, \alpha \rangle$  isomorphic to the input graph is created.  $A=\{A_1, A_2, \dots, A_n\}$  is the set of learning automata each of which assigned to a vertex of the input graph; specifically  $A_i$  is assigned to  $v_i$ ,  $\alpha=\{\alpha_1, \alpha_2, \dots, \alpha_n\}$  is the action-set in which  $\alpha_i = \{\alpha_i^1, \alpha_i^2, \dots, \alpha_i^{r_i}\}$  is the set of actions where  $r_i$  is the number of actions that can be taken by learning automata  $A_i$ . Action  $\alpha_i^j$  is the selection of vertex  $v_j$  by automaton  $A_i$ . Let  $p(v_i) = \{p_i^1, p_i^2, \dots, p_i^{r_i}\}$  be the action probability vector of learning automaton  $A_i$ , where  $p_i^j$  is initialized to  $1/r_i$  for all  $j$ . Let  $\Phi_k$  be the candidate stochastic vertex cover at iteration  $k$  and initially set empty. At first all learning automata are in inactive mode. Also a new vertex  $v_t$  is added to the input graph and then connected to all vertices of the input graph. A pure chance learning automaton with  $n$  actions each of which correspond to one of the vertices of the input graph will be assigned to this vertex. A pure chance learning automaton is a learning automaton which always selects its actions with equal probability (Narendra and Thathachar 1989).

The insertion of the new vertex  $v_t$  allows the algorithm to continue the traversal of the input graph when it encounters situations like entering a node with degree one or two which causes the current iteration of the algorithm terminates and a new iteration with a new randomly selected starting vertex begins. This causes the information gathered during that iteration to be lost and results in prolonging the process of finding the solution by the algorithm

### 3.2. Construction of stochastic vertex cover

In this step, a starting vertex called  $v_i$  is randomly selected from one of the vertices of the input graph. Then  $v_i$  is inserted into candidate stochastic vertex cover  $\Phi_k$ , the neighbors of  $v_i$  are inserted into  $N_k$  and learning automata  $A_i$  corresponding to  $v_i$  is activated. Learning automaton  $A_i$  chooses one of its actions  $\alpha_i^j$  (selection of node  $v_j$  adjacent to vertex  $v_i$ ) according to its action probability vector. Each inactive learning automaton connected to  $A_i$  disables action  $\alpha_i^j$  from its action set and then scales its action probabilities according to equation (3) and also disables those actions that cause selection of the vertices belongs to the  $N_k$  except  $v_i$  and scale its action probabilities. After that, automaton  $A_j$  corresponding to vertex  $v_j$  is activated and selects an action. Let this action be selecting vertex  $v_l$ . If selected vertex  $v_l$  is not  $v_t$ , then vertex  $v_l$  will be added to the candidate stochastic vertex cover  $\Phi_k$  and their new neighboring vertices to  $N_k$ . The process of activating a learning automaton, choosing an action, changing the number of actions for inactive automata connected to the activated automaton and neighbors  $N_k$ , adding a vertex to candidate vertex cover  $\Phi_k$  and its neighbors to  $N_k$  is repeated until the algorithm cannot activate another automaton. The algorithm may not be able to activate an automaton if either  $\Phi_k$  is a vertex cover or action set of the activated automaton becomes empty.

### 3.3. Computation of dynamic threshold

Dynamic threshold is used by the algorithm to judge the quality of current solution found by the algorithm. Dynamic threshold at iteration  $k$ ,  $DT(k)$ , which is the average of the weights of all stochastic vertex cover found so far is computed according to the following equation.

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

where  $k$  is the number of iteration,  $\bar{w}(\Phi_i)$  is the sum of the weights of the vertices in  $\Phi_i$ . Dynamic threshold is used to decide whether or not a vertex cover found during an iteration of the algorithm can become a candidate for stochastic minimum vertex cover. If the vertex cover found during iteration  $k$  has a weight

equal or lower than the dynamic threshold  $DT(k)$  then that vertex cover becomes a candidate for the stochastic minimum vertex cover  $\Phi^*$ .

### 3.4. Updating action probabilities

In this step, the probability vectors of learning automata assigned to the visited vertices of the input graph are updated according to the learning algorithm on the basis of the results obtained from the previous steps. If  $\bar{w}(\Phi_k)$  is lower than or equal to dynamic threshold  $DT(k)$ , then the actions selected by all automata along the traversed path are rewarded and penalized otherwise. At the end of this step, the disabled actions from the action set of activated learning automata will be enabled and their action probability vectors are rescaled according to equation (4) to reflect enabling the disabled actions. Finally, all activated learning automata will become inactive.

### 3.5. Stopping the algorithm

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

$$PC(k) = \prod_{v_i \in V, v_j \in \Phi_k} (\max(p_i^j)) \quad (6)$$

$PC(k)$  is the product of maximum probabilities of probability vectors of learning automata of the vertices connected directly to the vertices of candidate stochastic vertex cover  $\Phi_k$  at iteration  $k$ .  $p_i^j$  is the action probability vector of learning automaton  $A_i$  residing in vertex  $v_i$  which is connected to  $v_j$ .

Figure 2 shows the pseudo code for the proposed Algorithm 1.

<b>Algorithm 1:</b> The proposed algorithm for finding stochastic minimum vertex cover	
<b>Input:</b>	Stochastic Graph $G(V, E, W)$ , Thresholds $P_{max}, K_{max}$
<b>Output:</b>	The stochastic minimum vertex cover $\Phi^*$
<b>Assumptions:</b>	<ul style="list-style-type: none"> <li>Create a network of learning automata isomorphic to graph <math>G</math> by assigning an automaton <math>A_i</math> to each vertex <math>v_i</math>;</li> <li>Insert a new vertex <math>v_i</math> to graph <math>G</math> and connect it to all other vertices and assign a pure chance learning automata to vertex <math>v_i</math>;</li> <li>Let <math>k</math> denotes the iteration number and initially set to 1;</li> <li>Let <math>\Phi_k</math> denotes the candidate stochastic minimum vertex cover found at iteration <math>k</math>;</li> <li>Let <math>N_k</math> denotes the neighboring vertices of <math>\Phi_k</math>;</li> <li>Let <math>\bar{w}(\Phi_k)</math> denotes the average weight of all samples taken from all vertices in <math>\Phi_k</math>;</li> <li>Let <math>\Phi^*</math> denotes the stochastic minimum vertex cover and initially is empty;</li> </ul>
<b>Begin</b>	<ul style="list-style-type: none"> <li>Let <math>DT(k)</math> denotes the dynamic threshold at iteration <math>k</math> and initially set to 0;</li> <li>Let <math>PC(k)</math> is the product of maximum probabilities of learning automata of the vertices connected directly to the vertices in <math>\Phi_k</math> at iteration <math>k</math>;</li> <li>Inactive all learning automata;</li> </ul>
<b>Repeat</b>	<ul style="list-style-type: none"> <li>Select <math>v_i</math> randomly as starting vertex;</li> <li><math>\Phi_k \leftarrow v_i</math>;</li> <li>Insert neighboring vertices of <math>v_i</math> into <math>N_k</math></li> <li><b>While</b> (a learning automaton can be activated) <b>Do</b> <ul style="list-style-type: none"> <li>Learning automaton <math>A_i</math> is activated and then chooses an action according to its action probability vector;</li> <li>Let the action chosen by <math>A_i</math> be the action corresponding to vertex <math>v_j</math>;</li> <li>Take a sample from vertex <math>v_j</math>;</li> <li>Disable the action corresponding to selecting vertex <math>v_i</math> and vertex set <math>N_k</math> and then scale their action probabilities according to equation (3);</li> <li>Learning automaton <math>A_j</math> is activated and then selects an action using its action probability vector;</li> <li>Let the action selected by <math>A_j</math> be selecting vertex <math>v_l</math>;</li> <li>Take a sample from vertex <math>v_l</math>;</li> </ul> </li> <li><b>If</b> (<math>v_l \neq v_i</math>) <b>then</b> <ul style="list-style-type: none"> <li><math>\Phi_k \leftarrow \Phi_k \cup v_l</math>;</li> <li>Insert neighboring vertices of <math>v_l</math> into <math>N_k</math>;</li> </ul> </li> </ul>

```

End if
Disable the action corresponding to selecting vertex  $v_j$  and vertex set  $N_k$  and then scale their action probabilities according to equation (3);
 $v_j \leftarrow v_l;$ 
End While
Compute the weight of stochastic vertex cover  $\bar{W}(\Phi_k)$ ;
Compute dynamic threshold  $DT(k)$  using equation (5);
If ( $\bar{W}(\Phi_k) \geq DT(k)$ ) Then
    Reward the chosen actions by all the activated learning automata;
Else
    Penalize the actions chosen by all the activated learning automata;
End If
Calculate  $PC(k)$  using equation (6);
Enable actions of all learning automata and rescale their action probability vector according to equation (4);
Deactivate all learning automata;
If ( $\bar{W}(\Phi_k) \leq \bar{W}(\Phi^*)$ ) Then
     $\Phi^* \leftarrow \Phi_k$  ;
End If
 $k \leftarrow k + 1$ ;
Until ( $PC(k)$  is greater than  $P_{max}$  or  $k$  is greater than  $K_{max}$ )
End Algorithm

```

**Figure 2.** Pseudo code for algorithm 1.

### 3.6. Improvements

In the remaining part of this section three improvements of the proposed algorithm are described. These modifications try to improve either the speed of the algorithm or the quality of solution.

#### 3.6.1. Improvement 1

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 treats all the traversed vertices equally during the traversal of the graph. For this improvement, we use  $a_i(k) = c \cdot d_i^j(k)$  as the learning parameter of automaton  $A_i$  at iteration  $k$  where  $c$  is a constant value between 0 and 1,  $d_i^j(k)$  is an estimation of reward probability for action  $\alpha_i^j$  computed by the algorithm at iteration  $k$ .  $d_i^j(k)$  is computed by dividing the number of times that action  $\alpha_i^j$  is selected up to time  $k$  by automaton  $A_i$  and received reward by the number of time that automaton  $A_i$  is activated. In this way, the edges which were selected more frequently will be more probable to be selected in the future. As in the experiment, such a strategy will lead to higher rate of convergence and fewer samples from the graph. Algorithm 1 in which the learning rate is adaptive is called *Algorithm 2*.

#### 3.6.2. Improvement 2

Algorithm 1, uses learning automata with varying number of actions and stopping criterion based on  $PC(k)$  which is the product of maximum probabilities in probability vectors of learning automata of the vertices of the candidate stochastic vertex cover  $\Phi_k$ . This may slow down the process of finding the vertex cover. A remedy to this problem, it seems one way to speed up the learning process is to use a same action-set for all learning automata used by algorithm 1. In this improvement, we let the action-set of each learning automaton  $A_i$  contains two actions as  $\alpha_i = \{\alpha_i^1, \alpha_i^2\}$  where action  $\alpha_i^1$  is selecting  $v_i$  to be a member of vertex set  $\Phi_k$  and action  $\alpha_i^2$  is selecting  $v_i$  to be a member of neighboring set  $N_k$ . In this respect, we compute  $PC(k)$  as given by following equation.

$$PC(k) = \prod_{v_i \in C_k} (p(v_i)) \quad (7)$$

where  $p(v_i)$  is the action probability vector of learning automaton  $A_i$  residing in vertex  $v_i$ . Algorithm 1 in which the above improvement is called *Algorithm 3*.

### 3.6.3. Improvement 3

For this improvement, we assign two learning automata to each vertex of the graph. The first learning automaton is used as in algorithm 1 to guide the traversal process for finding the promising paths in order to reduce the number of samples taken from the graph and the second learning automaton at the same time tries to speed up the process of learning. The resulting algorithm is called *Algorithm 4*.

## 4. Experimental Results

In this section, to evaluate the performance of the proposed algorithms, we conducted number of experiments on synthetic stochastic graphs. The stochastic graphs used for the experimentations and their characteristics are given in Table 1. The first three graphs are well-known stochastic graphs which are borrowed from (Hutson and Shier 2006), with their weights of vertices being random variables. The other test graphs are synthetic stochastic graphs which are generated randomly. These graphs have  $n$  vertices (i.e.,  $n \in \{20-100\}$ ) where each pair of vertices are connected with probability  $p=0.2$  and vertex weights for these graph are random variables with exponential distributions whose mean are chosen randomly from set  $\lambda \in \{1,2,3,4,5\}$ .

**Table 1.** Stochastic graphs used in simulations

Graphs	Parameter of distribution	Number of Vertices	Number of Edges
Alex1-B	-	8	14
Alex2-B	-	9	15
Alex3-B	-	10	21
Synthetic graphs	$\lambda \in \{1,2,3,4,5\}$	$n \in \{20-100\}$	-

### 4.1. Experimental settings

All experiments are conducted 30 times and their averages of number of samples taken from graph (TNS) and percentage of runs converged to the minimum vertex cover (PRV) are reported. The stopping criteria of proposed algorithms is either the number of iteration reached 10,000 iterations or the value of  $PC(k)$  as given by equation (6) or (8) is greater than 0.90. In the following 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, 2 and 3, the learning rate  $a$  is set to 0.01 and for Algorithm 2, the parameter  $c$  is set to 0.2.

#### 4.2.1. Experiment I

This experiment is conducted to study the performance of the proposed algorithms (Algorithm 1 and its variants) and standard sampling method (SSM) for different confidence levels in terms of total number of samples taken from graph (TNS) and the percentage of runs converged to the maximum stochastic vertex cover (PRV). For Algorithm 1, Algorithm 3 and Algorithm 4 initial value for learning rate  $a$  is set to 0.01 and for Algorithm 2 in which the learning rate is adjusted adaptively, constant  $c$  is set to 0.2. The results of this experiment are given in Table 2 for *Alex1-B*, Table 3 for *Alex2-B* and Table 4 for *Alex3-B*. From the results we may conclude that among the proposed algorithms, Algorithm 4 requires fewer samples need to be taken from the graph and has the highest percentage of runs converged to the maximum stochastic vertex cover (PRV). Also, the results show that Algorithm 2 with adaptive learning rate works better than algorithm 1

with fixed learning rate in terms of the percentage of runs converged to the maximum stochastic vertex cover (PRV). Adaptation of learning rate results in higher number of samples need to be taken from graph.

**Table 2.** Number of samples taken from *Alex1-B* using the proposed algorithms and SSM with different confidence levels

confidence level	Algorithm 1		Algorithm 2		Algorithm 3		Algorithm 4		SSM
	TNS	PRV	TNS	PRV	TNS	PRV	TNS	PRV	TNS
0.50	863	95	1631	95	1177	97	<b>385</b>	<b>100</b>	3171
0.60	1257	96	1821	96	1485	98	<b>490</b>	<b>100</b>	3187
0.70	1634	96	2278	97	1871	98	<b>854</b>	<b>100</b>	3118
0.80	2246	97	2563	98	1948	98	<b>1066</b>	<b>100</b>	3152
0.90	2749	98	2757	99	2312	99	<b>1212</b>	<b>100</b>	3404
0.95	2961	99	2950	<b>100</b>	2536	<b>100</b>	<b>1534</b>	<b>100</b>	3445

**Table 3.** Number of samples taken from *Alex2-B* using the proposed algorithms and SSM with different confidence levels

confidence level	Algorithm 1		Algorithm 2		Algorithm 3		Algorithm 4		SSM
	TNS	PRV	TNS	PRV	TNS	PRV	TNS	PRV	TNS
0.50	886	95	1123	95	1042	95	<b>978</b>	<b>99</b>	3413
0.60	1125	96	1456	95	1347	96	<b>1146</b>	<b>100</b>	3437
0.70	1842	96	1940	97	1812	97	<b>1350</b>	<b>100</b>	3418
0.80	2173	96	2497	98	2148	98	<b>1542</b>	<b>100</b>	3464
0.90	2459	98	2781	99	2643	99	<b>1825</b>	<b>100</b>	3743
0.95	2714	99	2932	<b>100</b>	2824	<b>100</b>	<b>2214</b>	<b>100</b>	3773

**Table 4.** Number of samples taken from *Alex3-B* using the proposed algorithms and SSM with different confidence levels

confidence level	Algorithm 1		Algorithm 2		Algorithm 3		Algorithm 4		SSM
	TNS	PRV	TNS	PRV	TNS	PRV	TNS	PRV	TNS
0.50	702	97	1013	97	861	97	<b>283</b>	<b>100</b>	3802
0.60	1089	97	1220	98	1007	98	<b>366</b>	<b>100</b>	3867
0.70	1298	98	1425	98	1201	98	<b>412</b>	<b>100</b>	3887
0.80	1679	98	1681	99	1528	99	<b>679</b>	<b>100</b>	3953
0.90	1934	99	2203	99	1902	100	<b>797</b>	<b>100</b>	4201
0.95	2387	<b>100</b>	2679	<b>100</b>	2298	<b>100</b>	<b>904</b>	<b>100</b>	4214

In order to study the statistical significance between the proposed algorithms, several multi-comparison statistical tests (Friedman and Iman–Davenport) with 95% confidence level ( $\alpha=0.05$ ) are conducted (García et al. 2009). As a statistical analysis, Friedman’s test is first applied to obtain rankings. To obtain the adjusted  $p$ -values for each comparison between the control algorithm (the best-performing one) and the other algorithms, Bonferroni–Dunn (Bon-Dunn), Holm and Hochberg tests are conducted as post-hoc methods (if significant differences are detected). The rankings obtained by Friedman’s test are presented in Table 5. According to the results of statistical significance in

Table 6, one can conclude that Algorithm 4 is ranked first. The  $p$ -value computed by Friedman’s test is 6.4403E-11, which is below the significance interval of 95% ( $\alpha=0.05$ ). Thus, a significant difference exists among the observed results. Post-hoc methods (Bonferroni–Dunn, Holm and Hochberg tests) are also performed to obtain the adjusted  $p$ -values.

Table 6 shows the adjusted  $p$ -values of the Bonferroni–Dunn, Holm and Hochberg tests. The adjusted  $p$ -value in table 6 indicates that Algorithm 4 outperforms the other algorithms with  $\alpha=0.05$ .

**Table 5. Average ranking of Friedman’s test of comparison algorithms on the stochastic test networks.**

Test results	Algorithm 1	Algorithm 2	Algorithm 3	Algorithm 4	SSM
--------------	-------------	-------------	-------------	-------------	-----

<b>Average ranking</b>	3.0416	2.9861	2.8055	1.6666	5.0000
<b>Ranking</b>	4	3	2	1	5

**Table 6.** *p*-values of the comparing algorithms on the stochastic test networks.

<b>Algorithm 4 vs.</b>	<b>Test results</b>				
	Unadjusted <i>p</i> -value	<i>Bon-Dunn p</i> -value	<i>Holm p</i> -value	<i>Hochberg p</i> -value	<i>Hommel-p</i> -value
<b>SSM</b>	8.1565E-25	3.2626E-24	3.2626E-24	3.2626E-24	3.2626E-24
<b>Algorithm 1</b>	4.8753E-07	1.9501E-06	1.4626E-06	1.4626E-06	1.4626E-06
<b>Algorithm 2</b>	1.0497E-06	4.1988E-06	2.0994E-06	2.0994E-06	2.0994E-06
<b>Algorithm 3</b>	1.0945E-05	4.3781E-07	1.0945E-05	1.0945E-05	1.0945E-05

#### 4.2.2. Experiment II

In this experiment, we compare the number of samples required by the proposed algorithms in order to reach a particular accuracy and then compare it with the number of samples needed by the standard sampling method (SSM) to reach the same accuracy. In order to have a 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 (J. A. Torkestani and Meybodi 2012), 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  $\alpha_i$ . 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 vertices in graph, we need to build a confidence level  $1-\varepsilon_i$  for each vertex  $v_i$  such that  $\sum_{i=1}^n \varepsilon_i = \varepsilon$ . In this experiment, same confidence level  $1-\varepsilon_0$  is assumed for all vertices of the stochastic graph. The minimum required number of samples for each vertex 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  $\varepsilon$  varies from 0.4 to 0.1 with increment 0.1 (or convergence rate from 60% to 90%) for the proposed algorithms. Different synthetic stochastic graphs with size from 20 to 100 are used. Table 7 to Table 10 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 reported in Table 11 to compare each pair of results. In Table 11, 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: 1) For all proposed 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 different graph sizes and different confidence levels; 2) From Table 11, we may observe that Algorithm 4 and Algorithm 1 require lowest and highest numbers of samples respectively than other algorithms for all different graph size and different confidence levels; 3) According to statistical significance reported by Table 11, in terms of number of samples taken from the graphs, Algorithm 4, Algorithm 3, Algorithm 2 and Algorithm 1 are ranked 1, 2, 3 and 4, respectively.

**Table 7.** 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	SSM
20	2729	2258	738	<b>638</b>	4424
30	3643	2461	1704	<b>1310</b>	11707
40	4910	4568	2637	<b>2492</b>	21289
50	12558	9954	3726	<b>3455</b>	23810
60	15312	14533	5387	<b>4157</b>	35129
70	17332	15884	6592	<b>4684</b>	39975
80	19078	16380	6843	<b>5309</b>	49006

90	24398	20184	8311	<b>7154</b>	58792
100	27152	23592	9840	<b>7690</b>	68096

**Table 8.** 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	SSM
20	4210	3171	961	<b>846</b>	6714
30	6561	4072	2844	<b>2147</b>	17734
40	8367	7381	4168	<b>3405</b>	32244
50	15504	12516	7516	<b>5506</b>	36059
60	17681	16188	8359	<b>6827</b>	53146
70	21852	18052	9879	<b>7471</b>	60531
80	24127	20489	10262	<b>8209</b>	74205
90	29863	23748	12760	<b>9136</b>	89032
100	34698	26315	14830	<b>11542</b>	103042

**Table 9.** 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	SSM
20	5276	4708	2769	<b>2130</b>	10272
30	13647	7365	5684	<b>4083</b>	27158
40	19130	10309	6353	<b>5598</b>	49405
50	21114	18109	8659	<b>6248</b>	55253
60	23721	19352	14563	<b>10430</b>	81408
70	26737	23019	20051	<b>18974</b>	92703
80	34901	29018	26532	<b>21246</b>	113653
90	41663	37022	31576	<b>25305</b>	136409
100	67511	56176	45756	<b>39823</b>	157861

**Table 10.** Number of samples taken from different test graphs using the proposed algorithms and using SSM at confidence level 0.9

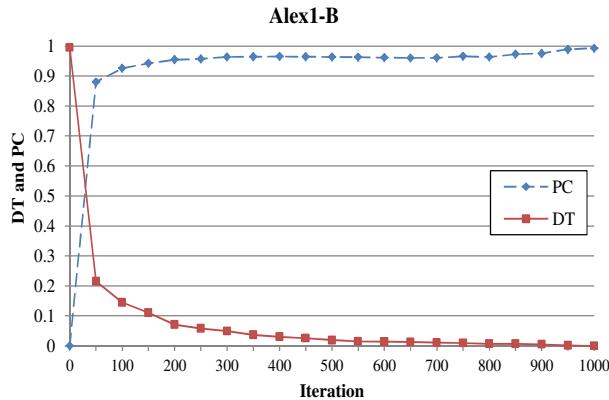
Graph size	Algorithm 1	Algorithm 2	Algorithm 3	Algorithm 4	SSM
20	7039	5923	5372	<b>3907</b>	16919
30	20381	10804	8205	<b>5742</b>	44725
40	31492	19159	9416	<b>7514</b>	81324
50	38265	26449	12988	<b>9831</b>	90971
60	45917	33452	19310	<b>16802</b>	134083
70	51021	39851	23018	<b>21739</b>	152692
80	67539	48611	32481	<b>27084</b>	187191
90	72887	52816	39163	<b>32709</b>	224583
100	88876	71958	43894	<b>36756</b>	259907

**Table 11.** The *t*-Test results of comparing algorithms in term of the number of samples taken from test graphs for different graph size

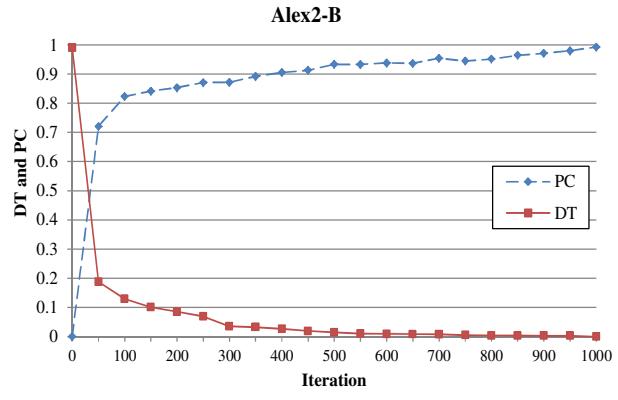
<i>t</i> -Test result \ Graph size	20	30	40	50	60	70	80	90	100
Algorithm 1 vs. Algorithm 2	<i>s-</i>	<b>-9</b>							
Algorithm 1 vs. Algorithm 3	<i>s-</i>	<b>-9</b>							
Algorithm 1 vs. Algorithm 4	<i>s-</i>	<b>-9</b>							
Algorithm 1 vs. SSM	<i>s+</i>								
Algorithm 2 vs. Algorithm 3	<i>s-</i>	<b>-9</b>							
Algorithm 2 vs. Algorithm 4	<i>s-</i>	<b>-9</b>							
Algorithm 2 vs. SSM	<i>s+</i>	<i>+9</i>							
Algorithm 3 vs. Algorithm 4	<i>s-</i>	<b>-9</b>							
Algorithm 3 vs. SSM	<i>s+</i>	<i>+9</i>							
Algorithm 4 vs. SSM	<i>s+</i>	<i>+9</i>							

### 4.2.3. Experiment III

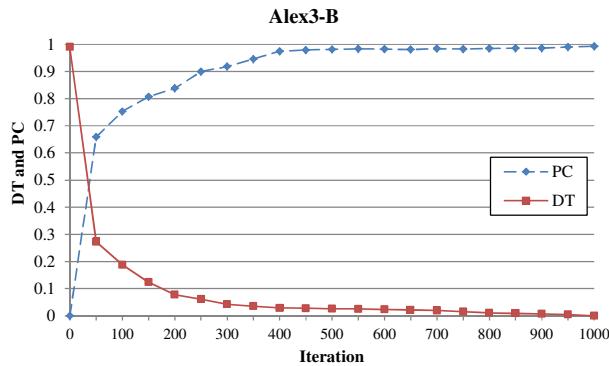
This experiment is conducted to study the behavior of Algorithm 4 during the process of finding the solution. In order to perform this experiment, we plot both  $PC(k)$  and  $DT(k)$  versus iteration number.  $DT(k)$  is scaled between (0,1) by dividing  $DT(k)$  by the weight of maximum vertex cover and subtraction of maximum vertex cover at the same time for each test network. Note that  $DT(k)$ , the dynamic threshold, is the average of the weights of all stochastic vertex cover found up to iteration  $k$  (equation (5)). The results of this experiment for stochastic test networks are given in Figure 3 through Figure 5. As it is shown,  $DT(k)$  gradually converges to the weight of minimum vertex cover and at the same time  $PC(k)$  converges to 1. This means that the algorithm gradually finds the vertex cover with minimum expected weight. Similar result can be obtained for other algorithms.



**Figure 3.** The plot of  $PC$  and  $DT$  versus iteration number for stochastic test graphs for Alex1-B



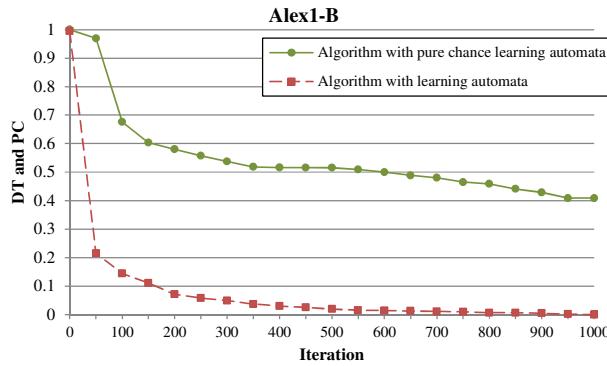
**Figure 4.** The plot of  $PC$  and  $DT$  versus iteration number for stochastic test graphs for Alex2-B



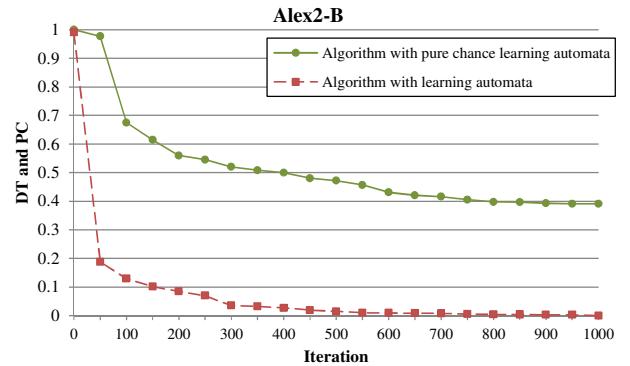
**Figure 5.** The plot of  $PC$  and  $DT$  versus iteration number for stochastic test graphs for Alex3-B

### 4.2.4. Experiment IV

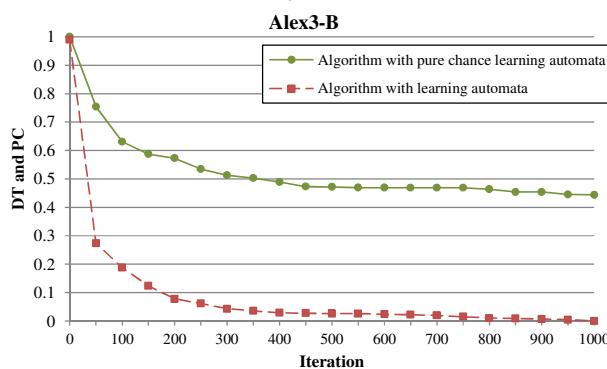
This experiment is conducted to compare algorithm 1 with algorithm 1 in which the leaning automaton residing in each vertex is replaced by a pure chance automaton. In pure chance automaton, the actions are selected with equal probabilities (Narendra and Thathachar 1989). The comparison is made with respect to  $DT$  scaled between (0, 1) by dividing  $DT$  by the weight of maximum vertex cover minus the weight of minimum vertex cover. The plot of  $DT$  versus iteration presented in Figure 6-10 show the role of learning automata in guiding the process of sampling from vertices of the stochastic graph for forming the stochastic minimum vertex cover. With the aid of learning automata the process of forming the stochastic minimum vertex cover 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.



**Figure 6.** Comparison of algorithm 1 with algorithm 1 in which learning automata are replaced with pure chance automata for Alex1-B



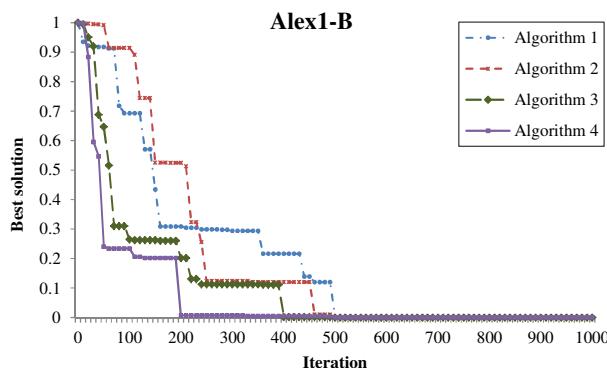
**Figure 7.** Comparison of algorithm 1 with algorithm 1 in which learning automata are replaced with pure chance automata for Alex2-B



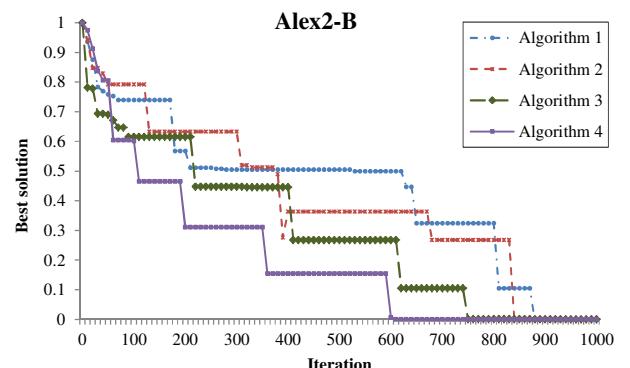
**Figure 8.** Comparison of algorithm 1 with algorithm 1 in which learning automata are replaced with pure chance automata for Alex3-B

#### 4.2.5. Experiment V

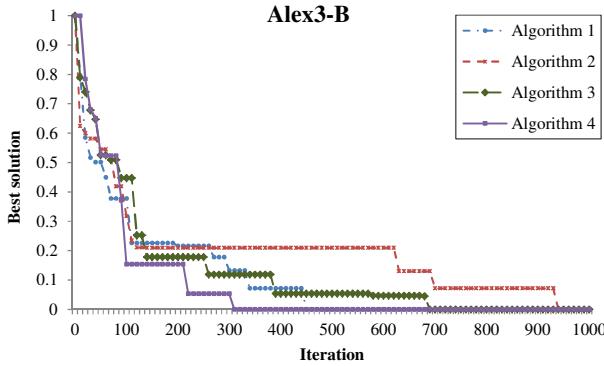
In this experiment, we study the convergence behavior of the proposed algorithms. For this purpose, we plot the weight of the best vertex cover found up to an iteration of the algorithms versus iteration number given in Figure 9-13. From the results one can conclude that Algorithm 4 has the highest convergence speed for all test graphs whereas Algorithm 1 and Algorithm 2 has the lowest speed of convergence. From Figure 9 we can also say that in terms of speed of convergence, Algorithm 4, Algorithm 3, Algorithm 2 and Algorithm 1 has rank 1, 2, 3 and 4, respectively.



**Figure 9.** Comparison of convergence behaviors of the proposed algorithms



**Figure 10.** Comparison of convergence behaviors of the proposed algorithms



**Figure 11.** Comparison of convergence behaviors of the proposed algorithms

## 5. CONCLUSION

In this paper, we was argued that because of uncertain, dynamic and time varying nature of real-world networks, stochastic graphs is a better model for such networks and then minimum vertex cover in stochastic graph was defined and four learning automata-based algorithms were proposed for solving minimum vertex cover problem in stochastic graphs under the condition that the probability distribution functions of the weights associated with the vertex 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 minimum vertex cover in stochastic graph as close to unity as possible. Experimental results showed that the proposed algorithms significantly reduce the number of samples required to be taken from the vertices of the stochastic graph as compared to the number of samples needed by the standard sampling method.

## Appendix I.

In this section, we describe 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 I.** Let  $q_i(k)$  be the probability of formation the vertex cover  $C_i$  at iteration  $k$ , and  $1-\varepsilon$  is the probability with which proposed algorithm 1 converges to the vertex cover  $\theta_i$ . 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

$$\frac{xa}{e^{xa} - 1} = \max_{j \neq i} \left( \frac{d_j}{d_i} \right) \quad (8)$$

Where  $1 - e^{-xa} = (1 - e^{-x})(1 - \varepsilon)$  and  $q_i = [q_i(k)|k = 0]$ .

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

$$V[u] = \begin{cases} \frac{e^u - 1}{e^u} & u \neq 0 \\ 1 & u = 0 \end{cases} \quad (9)$$

and also according to (Narendra and Thathachar 1989) we have

$$\frac{1}{V[x]} = \max_{j \neq i} \left( \frac{d_j}{d_i} \right) \quad (10)$$

It has been proved in [32,57] that there always exists a  $x > 0$  under which the equation (10) is satisfied, if  $\frac{d_j}{d_i} < 1$ , for all  $j \neq i$ . then we conclude that

$$\frac{e^{-xq_i/a} - 1}{e^{-x/a} - 1} \leq \Gamma_i(q) \leq \frac{1 - e^{-xq_i}}{1 - e^{-x}} \quad (11)$$

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

$$\Gamma_i(q) = \text{prob}[q_i(\infty) = 1 | q(0) = q] = \text{prob}[q^* = e_i | q(0) = q] \quad (12)$$

$q_i$  is the initial probability of the stochastic minimum vertex cover  $C_i$ . It is assumed that the probability with which the proposed algorithm 1 with learning rate  $a^*$  converges to the stochastic minimum vertex cover  $\theta_i$  is  $1-\varepsilon$ , for each  $0 < a < a^*$ , where  $a^*(\varepsilon) \in (0,1)$ . So it is concluded that,

$$\frac{1 - e^{-xq_i}}{1 - e^{-x}} = 1 - \varepsilon \quad (13)$$

It is also assumed that for every error parameter  $\varepsilon \in (0,1)$  there exists a value  $x$  under which the equation (10) is satisfied. So, we have

$$\frac{ax}{e^{ax} - 1} = \max_{j \neq i} \left( \frac{d_j}{d_i} \right) \quad (14)$$

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 minimum vertex cover 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 equation (14) 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 II.** To obtain a confidence level not smaller than  $1-\varepsilon$  for the minimum vertex cover, it is sufficient to build a confidence with level  $1-\varepsilon_i$  for every vertex  $v_i$  such that  $\sum_{i=1}^n \varepsilon_i = \varepsilon$ , where  $n$  is the number of vertices of vertex cover.

**Proof.** The required number of samples for each vertex of graph to satisfy a confidence level  $1-\varepsilon_i$  can be obtained using the standard method which described in section 2.6.

The problem is then to find a confidence region for each of vertices under which a desired confidence level  $1-\varepsilon$  is guaranteed for the minimum vertex cover. The confidence region for the minimum vertex cover is defined as the intersection  $\bigcap_{i=1}^n C_i(\varepsilon_i)$  of the confidence regions for the vertices, where  $C_i(\varepsilon_i) = 1 - \varepsilon_i$  is the confidence region for  $v_i$  and  $n$  is the number of vertices of minimum vertex cover. Using Booles-Bonferroni inequality (B Alt 1982), we have

$$\min_{1 \leq i \leq n} (1 - \varepsilon_i) > \text{prob} \left( \mu_i \in \bigcap_{i=1}^n C_i(\varepsilon_i) \right) \geq 1 - \sum_{i=1}^n p[\mu_i \notin C_i(\varepsilon_i)] \quad (15)$$

and so

$$\min_{1 \leq i \leq n} (1 - \varepsilon_i) > p \left( \mu_i \in \bigcap_{i=1}^n C_i(\varepsilon_i) \right) \geq 1 - \sum_{i=1}^n \varepsilon_i \quad (16)$$

Hence, the confidence level of the minimum vertex cover 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 minimum vertex cover. To achieve this, according to the *Bonferroni Correction* (Bonferroni 1936) it is sufficient to build a confidence with level  $1-\varepsilon_i$  for each vertex  $v_i$  such that  $\sum_{i=1}^n \varepsilon_i = \varepsilon$ , and hence the proof of the Theorem II.

## Appendix II.

Probability distribution function of small graphs (Alex1-B, Alex2-B, Alex3-B) (J. A. Torkestani and Meybodi 2010b) are listed as following tables.

**Table 12.** Stochastic Graph *Alex1-B*

Vertex	Weight	Probability
$v_1$	{2, 8, 12}	{0.90, 0.08, 0.02}
$v_2$	{10, 24, 35}	{0.85, 0.12, 0.03}
$v_3$	{6, 18, 24}	{0.88, 0.10, 0.02}
$v_4$	{12, 22, 30}	{0.85, 0.11, 0.04}
$v_5$	{17, 35, 50}	{0.75, 0.20, 0.05}
$v_6$	{3, 7, 10}	{0.68, 0.25, 0.07}
$v_7$	{4, 19, 15}	{0.75, 0.14, 0.11}
$v_8$	{5, 10, 12}	{0.65, 0.23, 0.12}

**Table 13.** Stochastic Graph *Alex2-B*

Vertex	Weight	Probability
$v_1$	{2, 8, 12}	{0.90, 0.08, 0.02}
$v_2$	{10, 24, 35}	{0.85, 0.12, 0.03}
$v_3$	{6, 18, 24}	{0.88, 0.10, 0.02}
$v_4$	{12, 22, 30}	{0.85, 0.11, 0.04}
$v_5$	{17, 35, 50}	{0.75, 0.20, 0.05}
$v_6$	{3, 7, 10}	{0.68, 0.25, 0.07}
$v_7$	{4, 19, 15}	{0.75, 0.14, 0.11}
$v_8$	{5, 10, 12}	{0.65, 0.23, 0.12}
$v_9$	{10, 19, 24}	{0.80, 0.14, 0.06}

**Table 14.** Stochastic Graph *Alex3-B*

Vertex	Weight	Probability
$v_1$	{2, 8, 12}	{0.90, 0.08, 0.02}
$v_2$	{10, 24, 35}	{0.85, 0.12, 0.03}
$v_3$	{6, 18, 24}	{0.88, 0.10, 0.02}
$v_4$	{12, 22, 30}	{0.85, 0.11, 0.04}
$v_5$	{17, 35, 50}	{0.75, 0.20, 0.05}
$v_6$	{3, 7, 10}	{0.68, 0.25, 0.07}
$v_7$	{4, 19, 15}	{0.75, 0.14, 0.11}
$v_8$	{5, 10, 12}	{0.65, 0.23, 0.12}
$v_9$	{10, 19, 24}	{0.80, 0.14, 0.06}
$v_{10}$	{18, 27, 36}	{0.94, 0.05, 0.01}

## 6. REFERENCES

- Akbari Torkestani, J. 2013. “A New Distributed Job Scheduling Algorithm for Grid Systems.” *Cybernetics and Systems* 44 (1): 77–93.  
Akbari Torkestani, J., and M. R. Meybodi. 2012. “A Learning Automata-Based Heuristic Algorithm for Solving the Minimum Spanning Tree Problem in Stochastic Graphs.” *The Journal of Supercomputing* 59 (2): 1035–54.

- B Alt, F. 1982. "Bonferroni Inequalities and Intervals." *Encyclopedia of Statistical Sciences*. <http://onlinelibrary.wiley.com/doi/10.1002/0471667196.ess0163/abstract>.
- Barabási, A. L., and R. Albert. 1999. "Emergence of Scaling in Random Networks." *Science* 286 (5439): 509–12.
- Beigy, H., and M. R. Meybodi. 2006. "Utilizing Distributed Learning Automata to Solve Stochastic Shortest Path Problems." *International Journal of Uncertainty Fuzziness And Knowledge Based Systems* 14 (5): 591.
- Bentz, C., M. -C. Costa, C. Picouleau, B. Ries, and D. de Werra. 2012. "D-Transversals of Stable Sets and Vertex Covers in Weighted Bipartite Graphs." *Journal of Discrete Algorithms*, Selected papers from the 2011 meeting of LSD/LAW, 17 (December): 95–102. doi:10.1016/j.jda.2012.06.002.
- Bonferroni, Carlo E. 1936. *Teoria Statistica Delle Classi E Calcolo Delle Probabilità*. Libreria internazionale Seeber.
- Boria, Nicolas, Federico Della Croce, and Vangelis Th. Paschos. 2014. "On the Max Min Vertex Cover Problem." *Discrete Applied Mathematics*. Accessed July 22. doi:10.1016/j.dam.2014.06.001.
- Bouamama, Salim, Christian Blum, and Abdellah Boukerram. 2012. "A Population-Based Iterated Greedy Algorithm for the Minimum Weight Vertex Cover Problem." *Applied Soft Computing* 12 (6): 1632–39.
- Cai, Shaowei, Kaile Su, Chuan Luo, and Abdul Sattar. 2013. "NuMVC: An Efficient Local Search Algorithm for Minimum Vertex Cover." *J. Artif. Int. Res.* 46 (1): 687–716.
- Cai, Shaowei, Kaile Su, and Abdul Sattar. 2011. "Local Search with Edge Weighting and Configuration Checking Heuristics for Minimum Vertex Cover." *Artificial Intelligence* 175 (9): 1672–96.
- Chvatal, Vasek. 1979. "A Greedy Heuristic for the Set-Covering Problem." *Mathematics of Operations Research* 4 (3): 233–35.
- Cuevas, E., D. Zaldivar, and M. Pérez-Cisneros. 2011. "Seeking Multi-Thresholds for Image Segmentation with Learning Automata." *Machine Vision and Applications* 22 (5): 805–18.
- Erdos, P., and A. Rényi. 1960. "On the Evolution of Random Graphs." *Publications of the Mathematical Institute of the Hungarian Academy of Sciences* 5: 17–61.
- Forsati, R., and M. R. Meybodi. 2010. "Effective Page Recommendation Algorithms Based on Distributed Learning Automata and Weighted Association Rules." *Expert Systems with Applications* 37 (2): 1316–30.
- García, Salvador, Daniel Molina, Manuel Lozano, and Francisco Herrera. 2009. "A Study on the Use of Non-Parametric Tests for Analyzing the Evolutionary Algorithms' Behaviour: A Case Study on the CEC'2005 Special Session on Real Parameter Optimization." *Journal of Heuristics* 15 (6): 617–44.
- Gupta, Anupam, Martin Pál, R. Ravi, and Amitabh Sinha. 2011. "Sampling and Cost-Sharing: Approximation Algorithms for Stochastic Optimization Problems." *SIAM Journal on Computing* 40 (5): 1361–1401.
- Gyarmati, László, and Tuan Anh Trinh. 2010. "Measuring User Behavior in Online Social Networks." *Network, IEEE* 24 (5): 26–31.
- Hasanzadeh, M., and Mohammad Reza Meybodi. 2014. "Grid Resource Discovery Based on Distributed Learning Automata." *Computing* 96 (9): 909–22.
- Hutson, K. R., and D. R. Shier. 2006. "Minimum Spanning Trees in Networks with Varying Edge Weights." *Annals of Operations Research* 146 (1): 3–18.
- Jadliwala, Murtuza, Igor Bilogrevic, and Jean-Pierre Hubaux. 2013. "Optimizing Mix-Zone Coverage in Pervasive Wireless Networks." *Journal of Computer Security* 21 (3): 317–46. doi:10.3233/JCS-130465.
- Jin-Hua, Zhao, and Zhou Hai-Jun. 2014. "Statistical Physics of Hard Combinatorial Optimization: Vertex Cover Problem." *Chinese Physics B* 23 (7): 078901. doi:10.1088/1674-1056/23/7/078901.
- Jin, Long, Yang Chen, Tianyi Wang, Pan Hui, and Athanasios V. Vasilakos. 2013. "Understanding User Behavior in Online Social Networks: A Survey." *IEEE Communications Magazine* 51 (9): 144–50.
- Jovanovic, Raka, and Milan Tuba. 2011. "An Ant Colony Optimization Algorithm with Improved Pheromone Correction Strategy for the Minimum Weight Vertex Cover Problem." *Applied Soft Computing* 11 (8): 5360–66.
- Jovanovic, Raka, Milan Tuba, and Dana Simian. 2010. "Comparison of Different Topologies for Island-Based Multi-Colony Ant Algorithms for the Minimum Weight Vertex Cover Problem." *WSEAS Transactions on Computers* 9 (1): 83–92.
- Karp, R.M. 1972. "Reducibility among Combinatorial Problems." *Complexity of Computer Computations* 40 (4): 85–103.
- Kempe, D., J. Kleinberg, and É Tardos. 2003. "Maximizing the Spread of Influence through a Social Network." In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 137–46.
- Kim, In-Jae, Brian P. Barthel, Yuyoung Park, Jordan R. Tait, Joseph L. Dobmeier, Sung Kim, and Dooyoung Shin. 2014. "Network Analysis for Active and Passive Propagation Models." *Networks* 63 (2): 160–69.
- Kuhn, Fabian, and Monaldo Mastrolilli. 2013. "Vertex Cover in Graphs with Locally Few Colors." *Information and Computation* 222: 265–77.

- Lakshmivarahan, S., and M. A. L. Thathachar. 1976. "Bounds on the Convergence Probabilities of Learning Automata." *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* 6 (11): 756–63.
- Liaqat, Hannan Bin, Feng Xia, Qiuyuan Yang, Zhenzhen Xu, Ahmedin Mohammed Ahmed, and Azizur Rahim. 2014. "Bio-Inspired Packet Dropping for Ad-Hoc Social Networks." *International Journal of Communication Systems*. <http://onlinelibrary.wiley.com/doi/10.1002/dac.2857/full>.
- Li, Lei, Xin Lin, Yue Zhai, Caixia Yuan, Yanquan Zhou, and Jiayin Qi. 2014. "User Communities and Contents Co-Ranking for User-Generated Content Quality Evaluation in Social Networks." *International Journal of Communication Systems*. <http://onlinelibrary.wiley.com/doi/10.1002/dac.2908/full>.
- Liu, Bing-Hong, Yu-Ping Hsu, and Wei-Chieh Ke. 2013. "Virus Infection Control in Online Social Networks Based on Probabilistic Communities." *International Journal of Communication Systems*. <http://onlinelibrary.wiley.com/doi/10.1002/dac.2630/full>.
- Maravall, D., and J. de Lope. 2011. "Fusion of Learning Automata Theory and Granular Inference Systems: ANLAGIS. Applications to Pattern Recognition and Machine Learning." *Neurocomputing* 74 (8): 1237–42.
- Misra, Sudip, and B. John Oommen. 2009. "An Efficient Pursuit Automata Approach for Estimating Stable All-Pairs Shortest Paths in Stochastic Network Environments." *International Journal of Communication Systems* 22 (4): 441–68.
- Moradabadi, Behnaz, and Hamid Beigy. 2014. "A New Real-Coded Bayesian Optimization Algorithm Based on a Team of Learning Automata for Continuous Optimization." *Genetic Programming and Evolvable Machines* 15 (2): 169–93.
- Morshedlou, H., and M.R. Meybodi. 2014. "Decreasing Impact of SLA Violations: A Proactive Resource Allocation Approach for Cloud Computing Environments." *IEEE Transactions on Cloud Computing* 2 (2): 156–67. doi:10.1109/TCC.2014.2305151.
- Mousavian, Aylin, Alireza Rezvanian, and Mohammad Reza Meybodi. 2013. "Solving Minimum Vertex Cover Problem Using Learning Automata." In *Proceedings of 13th Iranian Conference on Fuzzy Systems (IFSC 2013)*, 1–5. <http://arxiv.org/abs/1311.7215>.
- Najaflou, Y., B. Jedari, F. Xia, L.T. Yang, and M.S. Obaidat. 2014. "Safety Challenges and Solutions in Mobile Social Networks." *IEEE Systems Journal* in-press (doi: 10.1109/JST.2013.2284696): 1–21. doi:10.1109/JST.2013.2284696.
- Narendra, K. S., and M. A. L. Thathachar. 1989. *Learning Automata: An Introduction*. Prentice-Hall.
- Nicopolitidis, P. 2015. "Performance Fairness across Multiple Applications in Wireless Push Systems." *International Journal of Communication Systems* 28 (1): 161–66.
- Papoulis, A., and S. Pillai. 2002. *Probability, Random Variables and Stochastic Processes*. 4th ed. McGraw Hill.
- Rezvanian, A., Rahmati, M., and Meybodi, M. R. 2014. "Sampling from Complex Networks Using Distributed Learning Automata." *Physica A: Statistical Mechanics and Its Applications* 396: 224–34.
- Safavi, S. M., M. R. Meybodi, and M. Esnaashari. 2014. "Learning Automata Based Face-Aware Mobicast." *Wireless Personal Communications* 77 (3): 1923–33. doi:10.1007/s11277-014-1616-3.
- Shirali, M., N. Shirali, and M. R. Meybodi. 2012. "Sleep-Based Topology Control in the Ad Hoc Networks by Using Fitness Aware Learning Automata." *Computers & Mathematics with Applications* 64: 137–46.
- Singh, Alok, and Ashok Kumar Gupta. 2006. "A Hybrid Heuristic for the Minimum Weight Vertex Cover Problem." *Asia-Pacific Journal of Operational Research* 23 (02): 273–85.
- Soleimani-Pour, M., A. Rezvanian, and M. R. Meybodi. 2012. "Solving Maximum Clique Problem in Stochastic Graphs Using Learning Automata." In *2012 Fourth International Conference on Computational Aspects of Social Networks (CASoN)*, 115–19.
- Thathachar, M. 1987. "Learning Automata with Changing Number of Actions." *IEEE Transactions on Systems, Man, and Cybernetics* 17 (6): 1095–1100.
- Akbari Torkestani, J. 2012a. "Degree-Constrained Minimum Spanning Tree Problem in Stochastic Graph." *Cybernetics and Systems* 43 (1): 1–21.
- . 2012b. "An Adaptive Learning Automata-Based Ranking Function Discovery Algorithm." *Journal of Intelligent Information Systems* 39 (2): 441–59. doi:10.1007/s10844-012-0197-4.
- Akbari Torkestani, J. and M. R. Meybodi. 2010a. "Clustering the Wireless Ad Hoc Networks: A Distributed Learning Automata Approach." *Journal of Parallel and Distributed Computing* 70 (4): 394–405.
- . 2010b. "Learning Automata-Based Algorithms for Finding Minimum Weakly Connected Dominating Set in Stochastic Graphs." *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 18 (06): 721–58.
- . 2012. "Finding Minimum Weight Connected Dominating Set in Stochastic Graph Based on Learning Automata." *Information Sciences* 200 (1): 57–77.

- Akbari Torkestani, J. 2013. "A New Approach to the Vertex Coloring Problem." *Cybernetics and Systems* 44 (5): 444–66.
- Veremyev, Alexander, Alexey Sorokin, Vladimir Boginski, and Eduardo L. Pasiliao. 2014. "Minimum Vertex Cover Problem for Coupled Interdependent Networks with Cascading Failures." *European Journal of Operational Research* 232 (3): 499–511.
- Vos s, Stefan, and Andreas Fink. 2012. "A Hybridized Tabu Search Approach for the Minimum Weight Vertex Cover Problem." *Journal of Heuristics* 18 (6): 869–76.
- Watts, D.J., and S.H. Strogatz. 1998. "Collective Dynamics of 'small-World' Networks." *Nature* 393 (6684): 440–42.
- Yang, Kun, Xueqi Cheng, Liang Hu, and Jianming Zhang. 2012. "Mobile Social Networks: State-of-the-Art and a New Vision." *International Journal of Communication Systems* 25 (10): 1245–59.
- Yu, S. J. 2012. "The Dynamic Competitive Recommendation Algorithm in Social Network Services." *Information Sciences* 187: 1–14.
- Zhang, Zhao, Weili Wu, Lidan Fan, and Ding-Zhu Du. 2014. "Minimum Vertex Cover in Ball Graphs through Local Search." *Journal of Global Optimization* 59 (2-3): 663–71. doi:10.1007/s10898-013-0116-4.