Sampling algorithms for stochastic graphs: A learning automata approach

Alireza Rezvanian\textsuperscript{a,b,*}, Mohammad Reza Meybodi\textsuperscript{b}

\textsuperscript{a}School of Computer Science, Institute for Research in Fundamental Sciences (IPM), P.O. Box 19395-5746, Tehran, Iran
\textsuperscript{b}Soft Computing Laboratory, Computer Engineering and Information Technology Department, Amirkabir University of Technology (Tehran Polytechnic), Hafez Ave., 424, Tehran, Iran

\textbf{A R T I C L E   I N F O}

Article history:
Received 1 November 2016
Revised 22 April 2017
Accepted 24 April 2017
Available online 25 April 2017

Keywords:
Stochastic graphs
Network sampling
Network measures
Complex networks
Learning automata

\textbf{A B S T R A C T}

Recently, there has been growing interest in social network analysis. Graph models for social network analysis are usually assumed to be a deterministic graph with fixed weights for its edges or nodes. As activities of users in online social networks are changed with time, however, this assumption is too restrictive because of uncertainty, unpredictability and the time-varying nature of such real networks. The existing network measures and network sampling algorithms for complex social networks are designed basically for deterministic binary graphs with fixed weights. This results in loss of much of the information about the behavior of the network contained in its time-varying edge weights of network, such that is not an appropriate measure or sample for unveiling the important natural properties of the original network embedded in the varying edge weights. In this paper, we suggest that using stochastic graphs, in which weights associated with the edges are random variables, can be a suitable model for complex social network. Once the network model is chosen to be stochastic graphs, every aspect of the network such as path, clique, spanning tree, network measures and sampling algorithms should be treated stochastically. In particular, the network measures should be reformulated and new network sampling algorithms must be designed to reflect the stochastic nature of the network. In this paper, we first define some network measures for stochastic graphs, and then we propose four sampling algorithms based on learning automata for stochastic graphs. In order to study the performance of the proposed sampling algorithms, several experiments are conducted on real and synthetic stochastic graphs. The performances of these algorithms are studied in terms of Kolmogorov-Smirnov D statistics, relative error, Kendall’s rank correlation coefficient and relative cost.

© 2017 Elsevier B.V. All rights reserved.

1. Introduction

Many real-world complex phenomena in computer, biological, chemical, technological, information and social systems take the form of networks, which are represented as graphs with a set of nodes (e.g., users of social networks) and edges (e.g., a kind of relations between users of social networks). Numerious studies have been conducted to capture the structural and dynamical characteristics of complex social networks in various applications [1–4], while many real networks have found the small-world property [5], a power-law degree distribution property [6] or existence of community structures [7] in networks. The existing research aims to study and analyze of real-world networks, such as social networks reported in the literature, in order to demonstrate that the most of graph models are usually assumed to involve deterministic graphs with fixed weights for their edges or nodes, however the behavior or structure of social networks has an unpredictable, uncertain and time-varying nature, thus deterministic assumptions for graph models of online social networks may lose real information about the network due to the nondeterministic nature of real-world networks. For example in online social networks, the activity behaviors of online users such as following friends, liking comments, posting status posts, visiting user profiles, rating recommendations, asking questionnaires, the frequency of event notification, and the frequency of sharing news, to name a few, vary over time with unknown probabilities [8]. Analyzing online social networks with deterministic graph models do not take into consideration the continuum and variety of activities of the users occurring over time. Even modeling complex social networks with weighted graphs in which the edge or node weights are assumed to be fixed weights, only considers a snapshot of a network. Moreover, in analyzing online social networks not only understanding
the structure and topology of the network is important but also the dynamics and degree of association among the users in the network play a significant role in several analytical applications of online social networks.

In online social networks, understanding social relationships between users is relevant to any basic analysis that seeks to identify central/influential users and how they maximize the spread of influence through a social network. Central users, such as influential spreaders in the network may not necessarily be users with the maximum number of friends at any particular time; rather, the behavior of users over time is more important than the number of friends they have. For example, many users share their daily life activities in the form of photos, videos, and news or collaborate on a scientific paper with their friends or colleagues through online social network services. Sharing items and activities among audiences, such as in the form of their likes, comments, joining communities and making citations to a scientific paper, which may change over time, can impact on the behavior of other users such that they become interested in the favorite items of their friends or the work of collaborators. Indeed, in online social networks, the degree of friendship among the users changes directly with the weights of their edges to one another [9] over time, while it is clear that, at times, some edges with high weights are more important than those of certain others because they are more accessible and willing to communicate or take part in friendship activities. In contrast, there are times when the same users are on vacation, such that their contributions on their social graph have less edge weight because they are less accessible and have decreased their activities in online social networks.

Modeling real networks as deterministic binary networks with fixed weights simplifies the metrics, models, algorithms, applications and analyses involved. Furthermore, it results in intentionally losing much of the important information contained in the varying edge weights of the network, thereby reflecting the real nature of the network. A deterministic view about central users and their activities only focuses on the impact of the activities of users at a particular time and ignores the time-varying nature of such users’ activities and relations. As a solution to this problem, some researchers have recently suggested that, since understanding the characteristics of social relationships between users is the basis for capturing the realistic analysis of online social networks, stochastic graphs [9–13], in which the weights associated with the edges are random variables may be a better candidate for modeling complex social networks, as stochastic network measures which are random variables, provide more information for analyzing the network. Once the network model is chosen to take the form of stochastic graphs, every aspect of the network, such as path, clique, spanning tree, dominating set, network measures and sampling algorithms should be treated stochastically. For example, choosing a stochastic graph as the graph model of an online social network and defining a community structure [14] in terms of a clique [15], and the associations among the nodes within the community as random variables, the concept of a stochastic clique may be used to study community structure properties. Another example can involve identifying influential nodes with respect to network centrality measures, such that the concept of stochastic network centrality measures may be used to study the influence maximization by identifying influential nodes in stochastic graphs. Recently, several studies regarding modeling social networks as stochastic graphs have been reported in the literature. In [9,12,13], some network problems such as maximum clique [9], shortest paths [16] and minimum vertex covering [13] in stochastic graphs, were discussed. In particular, network measures and sampling algorithms must be redefined in order to deal with stochastic graphs directly, so as to provide useful and real analysis of social networks.

Studying, characterizing and analyzing of complex networks such as online social networks, can be realized by network global and local measures (e.g., diameter, average path length, average degree, clustering coefficient, degree distribution), as well as network sampling. The network measures can be used for describing the status of users (local view of the network) or the status of communities in networks (global view of the network) with a single parameter. Network sampling can provide an energy-efficient way to conduct any research into complex social networks where there is limited time and with lower computational cost (e.g., process, storage, bandwidth, power). Network sampling aims to construct a representative sampled network on a small scale, which has preserved the properties of the original network. The properties of the networks can be studied and characterized by network sampling with some network measures, which are used to study the network, instead of assessing the whole network data [17–20]. The network sampling algorithms reported in the literature solely focus on deterministic binary graphs, which only consider the existence or absence of a connection between nodes with fixed weights associated with the edges. These algorithms are not suitable for sampling stochastic graphs because they are not able to reveal the important natural properties of the original network embedded in the time-varying edge weights reflecting the real nature of the network.

In this paper, we first redefine some of the network measures such as strength, clustering coefficient, closeness and betweenness, for stochastic graphs; then, using a network of learning automata, the generalization of four previously introduced network sampling algorithms [20–23] for sampling stochastic networks are presented. The proposed sampling algorithms based on the network of learning automata aim to estimate the unknown distribution of the weight associated with an edge by taking samples from that edge for this purpose, all the proposed sampling algorithms use a network of learning automata to guide the process of sampling the edges of the stochastic graph in such a way that the number of samples taken from the edges is reduced as much as possible. A learning automaton is an adaptive decision-making unit that improves its performance by learning how to choose the optimal action from a finite set of allowed actions through repeated interactions with a random unknown environment. In order to study the performance of the proposed sampling algorithms for stochastic graphs, several experiments are conducted on real and synthetic stochastic networks; then, using the presented stochastic network measures, the proposed sampling algorithms are compared with respect to the Kolmogorov-Smirnov D-statistics, skew divergence, relative error, the Kendall’s rank correlation coefficient and relative cost.

The rest of this paper is organized as follows. Section 2 as backgrounds and preliminaries introduces stochastic graphs, network measures for deterministic networks, network sampling and its related work and learning automata theory. In Section 3, the proposed network measures are redefined and also generalization of four sampling algorithms based on learning automata for stochastic graphs is described in Section 4. The performance of the proposed sampling algorithms for stochastic graphs is studied through the simulation experiments reported in Section 5, and finally Section 6 concludes this paper.

2. Background and preliminaries

In this section, we provide a brief introduction to stochastic graphs, network measures for deterministic networks, network sampling and its related work and learning automata theory.
2.1. Stochastic graphs

A weighted undirected graph can be described by a triple $G = (V, E, W)$ where $V = \{v_1, v_2, ..., v_n\}$ is the set of nodes, $E = \{e_{ij}\}_{e \in E} \times V$ is the edge set, in which $e_{ij}$ indicates a connection between node $v_i$ and node $v_j$, and $W$ is a matrix in which $w_{ij}$ is a weight associated with the edge between nodes $v_i$ and $v_j$, if such an edge exists. Graph $G$ is said to be stochastic if weight $w_{ij}$ associated with edge $e_{ij}$ is a random variable with the probability density function (PDF) $q_{ij}$.

Set of nodes $V$ can represent online users in social networks, authors in citation networks, friends in friendship networks or airline terminal networks. An edge in edge set $E$ can represent a type of connection, relationship, task, activity or flow, which can be generated, produced, transferred or terminated through the edges of a network. The weights may be defined according to the tasks/activities on an edge between two nodes, such as strength activities between two users in online social networks, that is not always deterministic: the number of collaborations (co-authoring a paper) between scientific authors may change over time, the level of trust between friends also varies with time, and the links in a communication network can be affected by collisions, congestions, interferences or other factors. In short, the weights of connections in real-world network activities for each mentioned process may vary over time. Hence, it seems better to represent these networks as stochastic graphs, where the weights associated with the nodes or edges are random variables, rather than deterministic graphs.

In recent years, due to the important role of edge weights and their changes over time in various applications of network sciences, several research studies have been conducted on the stochastic graphs [9, 13, 24-26]. Ashii et al. [27] first discussed the spanning tree problem where the edge weights are expressed as random variables, and presented a polynomial time algorithm to solve it when the parameters of probability distributions of the edge weights are unknown. Torkeastani et al. [24] designed a learning automata-based heuristic algorithm, which significantly decreases the rate of unnecessary samples in order to solve the stochastic spanning tree problem with unknown probability distributions of weights. Furthermore, in [26], the degree-constrained minimum spanning tree problem on a stochastic graph was considered, while in [28], the mobility-based multicast routing algorithm for wireless mobile Ad-hoc networks are modeled with a stochastic graph and a stochastic version of the minimum Steiner connected dominating set problem in weighted networks, in which the relative mobility of each host is considered as its weight is introduced. Rezvani et al. [9] presented several distributed learning automata methods for finding the maximum clique in stochastic graphs for use in social network analysis. They also developed multiple learning automata-based algorithms for finding the minimum vertex covering in stochastic graphs [13], motivated by information spreading throughout online social networks, and showed that their algorithm with two sets of learning automata significantly reduces the number of samples required to be taken from the nodes of the stochastic graph compared to the number of samples needed by an algorithm with a single learning automaton and the standard sampling method. Moradabadi et al. modeled the time series link prediction problem in online social networks as a stochastic graph problem by concurrently considering different similarity metrics and link occurrences over time. Their solution is established based on a team of continuous action set of learning automata for solving a stochastic graph problem [29].

2.2. Network measures for deterministic networks

Network measures and computing them play a significant role in social network analysis [30]. Popular network measures, such as degree, betweenness, and closeness, are used not only in the characterization of a network, but also as part of computing some algorithms, such as the Girvan-Newman community detection algorithm [31] and overlapping community detection using nodes’ closeness [32]. In this section, some of the well-known network measures for deterministic networks are discussed.

2.2.1. Degree

Degree as a basic network measure has been widely used in many studies, with degree of node $v_i$ defined in deterministic binary graphs as follows:

$$D(v_i) = \sum_{j \neq i} a_{ij}$$  \hspace{1cm} (1)

where $j$ is the index of all other nodes of graph, and $a_{ij}$ is 1 if node $v_i$ is adjacent to node $v_j$, or 0 otherwise. In other words, degree of node $v_i$ counts the number of nodes that are directly connected to node $v_i$. Degree centrality is useful in the context of finding the single node that is affected by the diffusion of any information in the network. It follows from the fact that the node with a high degree of centrality has the chance to become affected by several source nodes [33].

2.2.2. Strength

The strength of a node is a weighted version of node degree, with strength of node $v_i$ is defined as the sum of adjacent weights as follows:

$$S(v_i) = \sum_{j \neq i} w_{ij}$$  \hspace{1cm} (2)

where $w_{ij}$ is greater than 0, if node $v_i$ is adjacent to node $v_j$ and its value indicates the weight of edge between node $v_i$ and node $v_j$. Similar to degree, a node with high-strength centrality is known as a popular node with high-strength edges to other nodes; however, a node with high strength may not necessarily consists of the maximum number of adjacent nodes. Strength centrality is useful in the context of finding a node that becomes affected by the amount of information that is spread throughout the network [33].

2.2.3. Closeness

The closeness of a node is the inverse sum of the shortest paths to all other nodes from that node and defined for deterministic graphs with $n$ nodes as follows:

$$CC(v_i) = \frac{1}{\sum_j d_{ij}}$$  \hspace{1cm} (3)

where $d_{ij}$ is the length of shortest path between node $v_i$ and node $v_j$. Closeness can be regarded as a measure of how long it will take to spread information from that node to all other nodes sequentially. Since the spread of information can be modeled by the use of the shortest paths, in applications, such as the spread of information, a node with high-closeness centrality can be considered as the central point because that node can spread the information faster than other nodes with lower-closeness centrality [33].

2.2.4. Betweenness

The betweenness of a node is the number of shortest paths from all nodes to all other nodes of the network that pass through that node. Betweenness of node $v_i$ is defined for deterministic graphs as follows:

$$B(v_i) = \frac{g_{st}(v_i)}{g_{st}}$$  \hspace{1cm} (4)

where $g_{st}$ is the number of shortest paths between all pairs of nodes of source node $v_i$ and destination node $v_i$ in network, while $g_{st}(v_i)$ is the number of those paths that pass through node $v_i$. 
Since a node with high-betweenness centrality will typically be the one acting as a bridge between many pairs of nodes, the betweenness is introduced as a measure for quantifying the control of a human regarding communication between other humans in a social network. In this conception, nodes with a high probability of occurrence on a randomly chosen shortest path between two randomly chosen nodes have a high betweenness. Betweenness can also be regarded as a measure of how to control information flows via communication links, while information flows can be dominated by nodes with high-betweenness centralities [33,34].

2.2.5. Clustering coefficient

The clustering coefficient for node $v_i$ is defined as a proportion of the number of all edges among its neighbors over the possible number of edges between them. The local clustering coefficient for a node $v_i$ can be defined as follows:

$$CC(v_i) = \frac{1}{k_i(k_i - 1)} \sum_{v_i,v_j \in N} a_{ij} a_{ih} a_{jh}$$

(5)

where $k_i = D(v_i)$ is the degree of node $v_i$, $N_i$ is the set of nodes (e.g., $v_i$ and $v_j$) adjacent to node $v_i$, and $a_{ij}$ is 1 if node $v_i$ is adjacent to node $v_j$, or 0 otherwise. The clustering coefficient of a node measures the connectivity among the neighborhood of the node (transitivity of a node). A node with a high clustering coefficient indicates the high tendency of that node to form a cluster with other nodes. In other words, the clustering coefficient measures the transitivity of a network. In addition, most of the neighbors of a node with a high clustering coefficient can collaborate with one another, even if the node is removed from the network.

2.3. Network sampling

Let $G = (V, E)$ be the original network, $V$ be the set of nodes and $E$ denotes the edge-set. Network sampling is a sampling method for constructing a representative sampled network $G' = (V', E' )$ from the original network $G$, such that $V' \subset V$ and $E' \subset E$ with sampling rate $0 < \phi < 1$, where $|V'| = \phi \times |V|$. It is noted that the main objective of network sampling is to achieve a small representative sampled network, while preserving the main properties of the original network $G$. Network sampling for stochastic graphs can be redefined as follows: let $G = (V, E, W)$ be the original network, $V$ be the set of nodes, $E$ denote the edge set and $W$ be the set of edge weights of the graph in which $w_{ij}$ associated with every edge $e_{ij} \in E$, is a random variable with unknown PDF $q_{ij}$. Network sampling for stochastic graphs is a sampling method for constructing a sampled network $G' = (V', E', W')$ from original network $G$, such that $V' \subset V$, $E' \subset E$ and $W'$ in which $w_{ij}'$ is an estimate of the edge weight associated with edge $e_{ij}' \in E'$ with sampling rate $0 < \phi < 1$, where $|V'| = \phi \times |V|.$

Due to the important role of network sampling algorithms for pre-processing, characterizing, studying and estimating the properties of online social networks, several sampling algorithms [17,21,23,35,36] have been presented for sampling networks. These can be classified into two groups: random selection sampling methods and network traversal sampling methods (also called link tracing sampling, traversal-based sampling, crawling-based sampling or topology-based sampling), such as distributed learning automata-based sampling (DLAS) [22], shortest path sampling (SSP) [21], spanning tree sampling (SST) [23] and extended distributed learning automata-based sampling (eDLAS) [20]. In the first groups, two simple techniques for random selection sampling are random edge sampling (RES) and random node sampling (RNS) [37], which are mainly used for theoretical investigations regardless of the topological structure of the networks involved. Network traversal sampling algorithms such as random walk (RW) [38], for-
automata \cite{52}. A Variable-Structure Learning Automaton (VSLA) is defined by a quadruple \((\alpha, \beta, p, \mathcal{T})\), where \(\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_r\}\) is the action-sets, \(\beta = \{\beta_1, \beta_2, \ldots, \beta_r\}\) denotes the set of inputs, \(p = \{p_1, p_2, \ldots, p_r\}\) is the probability of each action and \(p(t + 1) = \mathcal{T}[\alpha(t), \beta(t), p(t)]\) is the reinforcement scheme of automaton (also known as learning algorithm). At each instant \(t\), learning automaton randomly chooses an action \(\alpha(t)\) based on the action probability set \(p\), and performs it on the environment. After receiving the reinforcement signal \(\beta(k)\) from the environment, learning automaton updates its action probability set based on Eq. (6), if the chosen action \(\alpha_i(t)\) is rewarded by the random environment, and it is updated according to Eq. (7) if the taken action is penalized.

\[
p_j(t + 1) = \begin{cases} p_j(t) + a \left[ 1 - p_j(t) \right] & j = i \\ (1 - a) p_j(t) & \forall j \neq i \end{cases}
\]

where \(a\) is a reward parameter (also called learning rate) which determines the amount of increase of the action probability values, \(b\) is the penalty parameter determining the amount of decrease of the action probabilities values and \(r\) is the number of actions that learning automaton can take. For \(a = b\), the recurrence Eqs. (6) and (7) are called linear reward-penalty \((L_R,P)\) algorithm, for \(a > b\), they are called linear reward-\(\epsilon\)-penalty \((L_R,\epsilon)\) algorithm; and for \(b = 0\), they are called linear reward-inaction \((L_R,I)\) algorithm.

\[p_j(t + 1) = \begin{cases} (1 - b) p_j(t) & j = i \\ b \left[ \frac{1}{\mathcal{T}} \right] + (1 - b) p_j(t) & \forall j \neq i \end{cases}\]

\[\text{Fig. 1. Relationship between the learning automaton and its random environment.}\]

\[\text{Fig. 2. Distributed learning automata.}\]

\[\alpha(k)\]

\[\beta(k)\]

A distributed learning automata (DLA) \cite{53} is 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 LAs 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 \((A, E, T, A_0)\), 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 \(e_{ij}\) corresponds to action \(\alpha_j\) 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 chooses 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 nodes 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 by 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 Fig. 2, every automaton has two actions. If automaton \(A_i\) chooses \(\alpha_k\) from its action set, then it will activate automaton \(A_j\). Afterward, automaton \(A_k\) can choose one of its allowed actions and so on.

2.4.2. Extended distributed learning automata (eDLA)

An extended distributed learning automata (eDLA) \cite{16} is a new extension of DLA, which are supervised by a set of rules governing the operation of LAs. Mollakhalaji-Meybodi et al. presented a framework based on eDLA for solving stochastic graph optimization problems, such as the stochastic shortest path problem and stochastic minimum spanning tree problem. Here, we provide a brief introduction to the eDLA. In general, in eDLA, the ability of DLA is improved by adding communication rules and changing the activity level of each LA, depending on a problem to be solved by an eDLA. eDLA are similar to a DLA, in that they can be modeled by a directed graph in which the node set of a graph constructs the set of LAs, while the number of actions for each LA in eDLA equals the number of LAs that are connected to that particular LA. In eDLA, at any time, each LA can be in one mode of activity level, while each LA with a high activity level can perform an action according to its probabilities in the random environment. Formally, an eDLA can be described by a 7-tuple \((A, E, S, P, S_0, F, C)\), where \(A\) is the set of LA, \(E \subset A \times A\) is the edge set of communication graph \(G = (V, E)\) and \(S = \{s_1, s_2, \ldots, s_n\}\) is a set of activity levels corresponding to each LA in eDLA. Meanwhile \(s_i\) indicates the activity level for learning automaton \(A_i\), in which \(s_i \in \{P_a, A_c, Ac, Fi, Fo\}\) consists of one of the following activity levels: Passive (initial
level of each LA, which can be changed to Active), Active (activity level for a set of available LAs, which can be upgraded to Fire), Fire (the highest level of activity; in which the LA can be performed and its level can be changed to Off) and Off (the lowest level of activity, in which the LA is disabled and its level stays unchanged), represented briefly by Pa, Ac, Fi and Of respectively. As mentioned, at any time only one LA in eDLA can be in the Fi level of activity and determined by fire function \( C \), which randomly selects an LA from a set of LAs with an activity level of Ac. Governing rule \( P \) is the finite set of rules that governs the activity levels of each LA. P is defined according to the current activity level of each LA, its adjacent LA or the particular problem by which eDLA are designed. Defined \( S^0 = (s^0_1, s^0_2, \ldots, s^0_n) \) and \( F = \{S^0 \mid S^0 = (s^1_1, s^2_2, \ldots, s^n_1)\} \) the \( \text{initial state} \) and \( \text{final conditions} \) of eDLA.

The operation of eDLA can be described as follows. In eDLA, at first at initial state \( S^0 \), a starting LA is randomly selected by firing function \( F \) to fires, then selects one of outgoing edges (actions), according to its action probabilities, and performs it on the random environment. At the same time, the activity level of the fired LA and neighboring LAs are changed to Of and Ac, respectively. Changing activity levels of LAs results in the state of eDLA transferring from state \( S^k \) to state \( S^{k+1} \) at instant \( k \) by governing rule \( P \). Then firing function \( C \) fires one LA from the set of LAs with an activity level of Ac to select an action, after which it changes its activity level and neighboring LAs. The process of firing one LA by using the firing function, performing an action with the fired LA, and changing the activity level of the fired LA and its neighbors by governing rule \( P \), is continued until the final condition of eDLA \( F \) is reached. \( F \) can be defined based on a set of criteria in terms of activity levels of LAs, such that, if one of them is satisfied, the final condition of eDLA is realized. The environment evaluates the performed actions by the fired LAs and generates a reinforcement signal to eDLA. The action probabilities of the fired LAs along the visited nodes or the LAs of the nodes, which are part of a solution to the problem of the graph, are then updated on the basis of the reinforcement signal, according to the learning algorithm. Firing LAs of eDLA by starting with randomly selected LAs is repeated a predefined number of times until the solution to the problem for which eDLA is designed is obtained.

3. Proposed network measures for stochastic graphs

In this section, we first define some of the concepts in stochastic graphs such as path and spanning tree, and then define network measures, such as strength, closeness, betweenness and clustering coefficient, for the analysis of stochastic graphs.

**Definition 1** (stochastic graph). A stochastic graph \( G \) can be described by a triple \( G = (V,E,W) \), where \( V = \{v_1, v_2, \ldots, v_n\} \) is the set of nodes, \( E = \{e_{ij}\} \subseteq V \times V \) is the edge set, and \( W \) is a matrix, in which \( w_{ij} \) is a random variable associated with edge \( e_{ij} \) between node \( v_i \) and node \( v_j \), if such that edge exists.

**Definition 2** (stochastic path). In a stochastic graph, a path \( \pi \), with a weight of \( w_{\pi} \), and a length of \( n_{\pi} \) from source node \( v_1 \) to destination node \( v_n \), can be defined as an ordering \( \{\pi^1_{1}, \pi^1_{2}, \ldots, \pi^1_{n}\} \subseteq V \) of nodes in such a way that \( \pi_1^1 = v_1 \) and \( \pi_n^1 = v_n \) are source and destination nodes, respectively, and edge \( e(\pi^1_{j-1}, \pi^1_{j}) \in E \) for \( 1 \leq j \leq n \), where \( \pi_j \) is the \( j^{th} \) node in path \( \pi^1 \).

**Definition 3** (stochastic shortest path). Let, in a stochastic graph, there are \( r \) distinct paths \( \Pi = \{\pi_1^1, \pi_2^1, \ldots, \pi_r^1\} \) between source node \( v_1 \) and destination node \( v_n \), while path \( \pi^1 \) between source node \( v_1 \) and destination node \( v_n \) is defined as a stochastic shortest path if and only if \( w_{\pi^1} = \min_{\pi^1 \in \Pi} \{w_{\pi^1}\} \), where \( w_{\pi^1} = \sum_{e_{ik} \in \pi^1} w_{ik} \) is the expected weight of path \( \pi^1 \) and known as the stochastic shortest path, i.e., a stochastic shortest path is a stochastic path with the minimum expected weight.

**Definition 4** (minimum spanning tree). Let, in a stochastic graph \( G, \Theta = \{\tau^1, \ldots, \tau_r\} \) be the set of all possible spanning trees of graph \( G \) and \( w_{\pi^1} = \sum_{e_{ik} \in \pi^1} w_{ik} \) be the weight of spanning tree \( \tau^1 \), where \( w_{ik} \) is the edge weight of edge \( e_{ik} \in \Theta \). Therefore, spanning tree \( \tau^1 \) is the minimum spanning tree of graph \( G \) if and only if \( w_{\tau^1} = \min_{\pi^1 \in \Pi} \{w_{\pi^1}\} \).

**Definition 5** (stochastic minimum spanning tree). Let, in a stochastic graph, \( \Theta = \{\tau^1, \ldots, \tau_r\} \) be the set of spanning trees of a stochastic graph, in which stochastic spanning tree \( \tau^1 \) is called the stochastic minimum spanning tree if and only if \( w_{\tau^1} = \min_{\pi^1 \in \Pi} \{w_{\pi^1}\} \) where \( w_{\pi^1} \) is the expected weight of spanning tree \( \tau^1 \in \Theta \). That is the stochastic minimum spanning tree is a stochastic tree with the minimum expected weight.

**Definition 6** (Stochastic strength). The strength of node \( v_i \) in a stochastic graph is a random variable defined by following equation:

\[
SS(v_i) = \sum_{j \neq i} W_{ij}
\]

(8)

where \( w_{ij} \) is a random variable associated with the edge weight between node \( v_i \) and node \( v_j \).

**Definition 7** (stochastic closeness). The closeness of node \( v_i \) in a stochastic graph is a random variable defined by Eq. (9):

\[
SC(v_i) = \frac{1}{\sum_{j \neq i} ED_{ij}}
\]

(9)

where \( ED_{ij} \) is a random variable associated with the weight of the shortest path between node \( v_i \) and node \( v_j \) with a minimum expected weight.

**Definition 8** (stochastic betweenness). The betweenness of node \( v_i \) in a stochastic graph is a random variable \( SB(v_i) \) defined by Eq. (10):

\[
SB(v_i) = \frac{g_a(v_i)}{g_a}
\]

(10)

where \( g_a(v_i) \) is the distribution of the number of shortest paths between node \( v_i \) and node \( v_j \) that pass through node \( v_i \) while \( g_a \) is the distribution of the number of shortest paths between node \( v_i \) and node \( v_j \).

**Definition 9** (stochastic clustering coefficient). The clustering coefficient of node \( v_i \) in a stochastic graph is a random variable, which can be defined as follows:

\[
SCC(v_i) = \frac{1}{(D(v_i)-1)SS(v_i)} \sum_{j \neq i} \sum_{k \neq j} \sum_{l \neq j} \frac{(w_{ij} + w_{kl})}{2} a_{ij}a_{jk}a_{jl}
\]

(11)

where \( D(v_i) \) is the number of nodes adjacent to node \( v_i \) and \( a_{ij} = 1 \) if node \( v_i \) is adjacent to node \( v_j \), or \( 0 \) otherwise. \( SS(v_i) \) is the strength of node \( v_i \), while \( w_{ij} \) is the random variable associated with the weight of edge \( e_{ij} \).

Based on the proposed network concepts and measures in stochastic graphs, we propose four network sampling algorithms for stochastic graphs in the following section.

4. Proposed network sampling algorithms for stochastic graphs

In the previous section, we defined some network concepts and measures for stochastic graphs. In this section, we propose four network sampling algorithms for stochastic graphs. Let \( G(V, E, W) \) be the input stochastic graph, where \( V = \{v_1, v_2, \ldots, v_n\} \) is the set of nodes, \( E \) is the edge set and \( W = \{w_1, w_2, \ldots, w_m\} \) is the set
of random variables with unknown probability distribution functions, each of which is associated with an edge of the input graph. Each sampling algorithm tries to estimate the unknown distribution of the weight associated with an edge by taking samples from that edge. All the proposed sampling algorithms use a network of learning automata to guide the process of sampling from the edges of the stochastic graph in such a way that the number of samples taken from the edges is reduced as much as possible. These algorithms represent the generalization of four existing sampling algorithms reported for binary networks in [20–23].

4.1. Shortest path sampling for stochastic graphs (SPS-SG)

In this sub-section, we generalize the shortest path sampling (SPS) algorithm [21] for stochastic graphs, which we call it SPS-SG. The SPS-SG algorithm iteratively computes stochastic shortest paths between certain pairs of nodes (e.g., \( K_{\text{max}} \) pairs) using a network of learning automata and then uses them for constructing a sampled graph.

The SPS-SG uses a network of learning automata that isomorphic to the input graph by assigning automaton \( A_i \) to each node \( v_i \). An action of learning automaton \( A_i \), assigned to node \( v_i \), corresponds to the selection of one of the outgoing edges of node \( v_i \). Let \( \alpha_i = \{ \alpha_{i1}, \alpha_{i2}, \ldots \alpha_{in} \} \) be the set of actions for learning automaton \( A_i \), \( r_i \) be the number of outgoing edges from node \( v_i \), \( w_i \) be the average weight of edge \( e_{ij} \), \( \pi_i \) be the stochastic shortest path at duration stage \( t \), and \( L_s \) be a one-dimensional array with size \( |V| \), which is used to record the number of times that nodes are visited. In SPS-SG, at any time, each learning automaton can be in either an active or an inactive mode. Initially, all learning automata are set to inactive mode. SPS-SG consists of several stages. At each stage, the algorithm iteratively finds a stochastic shortest path between a randomly chosen pair of nodes, updates the action probability vector of learning automata until it finds a stochastic shortest path with minimum expected weight. The process of finding a stochastic shortest path with minimum expected weight at stage \( t \) can be described as follows.

At first, source node \( v_s \) and destination node \( v_d \), which are non-adjacent, are randomly selected from the set of nodes that are not yet selected as source or destination nodes. Then learning automaton \( A_{ij} \), corresponding to \( v_s \), is activated and chooses one of its actions according to its action probability vector. Let the chosen action be edge \( e_{ij} \). By performing this action, edge \( e_{ij} \) is visited and then inserted in \( \pi_i \), after which the average weight of the selected edge \( w_{ij} \) is recomputed and action \( \alpha_{ij} \) from the action set of each inactive learning automaton connected to \( A_i \) is disabled to avoid the loop. Learning automaton \( A_{ij} \) is then activated and chooses one of its actions, while the list of visited nodes \( L_s \) is updated. The process of activating a learning automaton, choosing an action, adding a visiting edge to shortest path \( \pi_i \) and updating the list of visited nodes \( L_s \) will be repeated until the algorithm reaches destination node \( v_d \) or the active learning automaton is unable to activate another automaton, which happens when all the actions of active learning automata are disabled. After that, the probability vectors of activated learning automata are updated according to the learning algorithm on the basis of the results obtained from the previous steps. The weight of stochastic shortest path \( \pi_i \) is compared with the average of the weights for all stochastic shortest paths found so far using the dynamic threshold, which is computed as \( T_t = (t-1)T_{t-1} + \Delta T / t \). If the weight of shortest path \( \pi_t \) is equal or lower than dynamic threshold \( T_t \), then the actions chosen by all the activated learning automata along path \( \pi_t \) are rewarded or otherwise penalized. At the end of this step, the disabled actions from the action set of activated learning automata will be enabled and all activated learning automata will become inactive. The process of finding the stochastic shortest path with minimum expected weight at this stage stops when the number of stage-number \( t \) exceeds a user-defined threshold \( T_{\text{max}} \) or when \( PC(t) = \prod_{v_i \in \pi_t} \left( \max(p(v_i)) \right) \) reaches a predefined threshold \( P_{\text{max}} \).

Finally, after finding \( K_{\text{max}} \) different stochastic shortest paths, the visited nodes are sorted according to the number of shortest paths along which that node has appeared. The sampled network can now be constructed by considering an induced subgraph of the input network, whose node set contains a given number of mostly visited nodes, while and its edge weights are the average weights of edges estimated during the execution of the algorithms. The pseudo-code of the SPS-SG as Algorithm 1 for sampling stochastic graph is given in Fig. 3.

4.2. Spanning trees sampling for stochastic graphs (SST-SG)

In this section, we propose another sampling algorithm for stochastic graphs, which is a generalization of the spanning tree sampling (SST) algorithm [23] for stochastic networks and call it SST-SG. This algorithm is the same as the SPS-SG except that, at stage of the algorithm, we use the concept of the stochastic minimum spanning tree instead of the concept of the stochastic shortest path. The SST-SG algorithm consists of several stages. At each stage, the algorithm finds a stochastic minimum spanning tree with a root randomly selected from the set of nodes of the input graph. After finding stochastic minimum spanning trees, the visited nodes, during all stages, are sorted according to the number of spanning trees along which the nodes have appeared. The sampled network can now be constructed by considering an induced subgraph of the input network whose node set contains a given number of mostly visited nodes, while its edge weights are the average weights of edges estimated during the execution of the algorithm. The pseudo-code of the SST-SG as Algorithm 2 for sampling stochastic graph is given in Fig. 4.

4.3. Distributed learning automata-based sampling for stochastic graphs

In this section, we generalize the distributed learning automata (DLAS) sampling algorithm [22] for used with sampling stochastic graphs, which we refer to as DLAS-SG. The DLAS-SG algorithm iteratively visits the nodes of the stochastic graph from different starting nodes on several times (e.g., \( K_{\text{max}} \) starting nodes) using a distributed learning automata, after which it uses the visited nodes to construct a sampled graph. The DLAS used for this purpose is a DLA that is isomorphic with the input graph. The action set of learning automaton \( A_i \), which is assigned to node \( v_i \), is the set of outgoing edges of node \( v_i \) on the input graph. Each learning automaton in DLA initially chooses its actions with equal probabilities.

DLAS-SG consists of number of stages. In each stage, a path in the graph is traversed and the average weights of the edges along this path are updated. To do this, DLA start from a randomly chosen starting node \( v_s \), while learning automaton \( A_i \) corresponding to node \( v_i \) is activated and chooses one of its actions according to its action probability vector. Let the chosen edge be \( e_{ij} \). The chosen edge \( e_{ij} \) is sampled (visited), then the average weight of the sampled edge is updated. Next, learning automaton \( A_j \) is activated and chooses one of its actions. The process of activating a learning automaton, choosing an action, activating another learning automaton and updating the average weight of the chosen edge is repeated until either the number of visited nodes reaches \( |V| \times \Phi \) or the activated learning automaton cannot activate another automaton. Once the traversal of path \( \pi_t \) at stage \( t \) is terminated, the probability vectors of learning automata along the traversed
path $\pi_t$ are updated as follows. If the weight of $\pi_t$ ($\bar{w}_{\pi_t}$) is equal or greater than dynamic threshold $T_t$, which is computed as $T_t = [(t-1)T_{t-1} + \bar{w}_{\pi_t}] / t$, then the probability of actions chosen by all the activated learning automata along the traversed path $\pi_t$ is increased, or decreased otherwise, according to the learning algorithm. At the end of a stage, all learning automata in DLA are deactivated, after which a new stage begins, provided that the maximum number of stages ($K_{max}$) is not reached and the difference between two dynamic thresholds in two consecutive stages is equal or greater than predefined threshold $T_{min}$.

When the execution of stages is over, the nodes visited during all stages are sorted in descending order, according to the number of paths along which a node has appeared. The sampled network is then constructed by considering an induced subgraph of the input network, whose node set contains a given number of mostly visited nodes and its edge weights are the average weights of edges estimated during the execution of the algorithms. The pseudo-code of the DLAS-SG as Algorithm 3 for stochastic graphs is given in Fig. 5.

4.4. Extended distributed learning automata-based sampling for stochastic graphs

In this section, we generalize the extended distributed learning automata (eDLAS) sampling algorithm [20] for sampling stochastic graphs, which we call it eDLAS-SG. The eDLAS-SG algorithm iteratively traverses stochastic graphs from different starting nodes (e.g., $K_{max}$ starting nodes) using eDLAS and then uses them to construct the sampled graph. The algorithm uses an eDLA that is isomorphic with the input graph. The action-set of learning automaton

---

**Algorithm 1. SPS-SG($G, \varphi, P_{max}, T_{max}, K_{max}$)**

**Input:** Stochastic graph $G=\langle V, E, W \rangle$, Sampling rate $\varphi$, Thresholds $P_{max}$, $T_{max}$, $K_{max}$.

**Output:** Sampled stochastic graph $G'=\langle V', E', W' \rangle$.

**Initialization**

Create a network of learning automata isomorphic to graph $G$ by assigning an automaton $A_t$ to each node $v_t$.

- $A_t$ is used to denote the stochastic shortest path between source node $v_t$ and destination node $v_{tj}$ during stage $t$.
- $\bar{w}_{v_t}$ denotes the average weight of all samples taken from edges along the path of $v_t$.
- $\pi'$ keeps track of the best path found so far during a stage of the algorithm.
- $Ls$ is one-dimensional array with size $|V|$ which is used to store the number of times that the nodes are sampled.

$NV$ is the set of nodes which are not selected yet as source or destination nodes, this set is initially set to $V$.

**Begin algorithm**

$k \leftarrow 1$

While ($k < K_{max}$)

- $t \leftarrow 1$, $T_t \leftarrow 0$, $\pi' \leftarrow \{\}$
- Inactive all learning automata;

**Repeat**

- Select two non-adjacent $v_t$ and $v_{tj}$ from non-selected nodes $NV$ randomly as source and destination nodes;
- $NV \leftarrow NV \setminus v_t$;
- Set $A_t$ to $A_{tj}$;

**While** ($v_t$ is not reached AND a learning automaton can be activated) **Do**

- Learning automaton $A_t$ is activated and then chooses an action according to its action probability vector;
- Let the action chosen by $A_t$ be edge $e_{tj}$;
- Take a sample from edge $e_{tj}$;
- $\pi_t \leftarrow \pi_t \cup \{e_{tj}\}$;
- $\bar{w}_{v_t} \leftarrow \bar{w}_{v_t} + \bar{w}_{e_{tj}}$;
- $Ls[v_t] \leftarrow Ls[v_t] + 1$;
- Disable action $e_{tj}$ from action set of each inactive learning automaton connected to $A_t$ to avoid the loop;
- Set $A_t$ to $A_{tj}$;

**End while**

- $T_t \leftarrow [(t-1)T_{t-1} + \bar{w}_{v_t}] / t$;
- If ($\bar{w}_{v_t} \leq T_t$) Then
  - Reward the chosen actions by all the activated learning automata along path $\pi_t$;
- Else
  - Penalize the chosen actions by all the activated learning automata along path $\pi_t$;

**End if**

- If ($\bar{w}_{v_t} \leq \bar{w}_{e}$) Then
  - $\pi' \leftarrow \pi_t$;
- End if

- $PC(t) \leftarrow \Pi_{e \in E}(\max(p(v_t)))$;
- $t \leftarrow t + 1$;
- Until ($PC(t) \geq P_{max}$ OR $t \geq T_{max}$)
- $k \leftarrow k + 1$;
- Enable all the disabled actions;

**End While**

- Sort $Ls$ in descending order;
- Construct an induced sub-graph $G'=\langle V', E', W' \rangle$ using $\varphi|V|$ mostly visited nodes;

**End Algorithm**

---

**Fig. 3.** The pseudo-code of the shortest path sampling algorithm for stochastic graphs (SPS-SG).
Algorithm 2. SST-SG(G, ϕ, P_{max}, T_{max}, K_{max})

Input: Stochastic graph \( G\equiv(V, E, W)\), Sampling rate \( \phi \), Thresholds \( P_{\text{max}}, T_{\text{max}}, K_{\text{max}} \).
Output: Sampled stochastic graph \( G'\equiv(V', E', W')\).

Initialization
Create a network of learning automata isomorphic to graph \( G \) by assigning an automaton \( A_i \) to each node \( v_i \).
\( \tau_i \) denotes the minimum stochastic spanning tree found at iteration \( t \).
\( \bar{w}_t \) denotes the average weight of all samples taken from edges along tree \( \tau_t \).
\( t \) keeps track of the best spanning tree found so far during a stage of the algorithm.
\( L_\text{s} \) is a one-dimensional array with size \( |V| \) which is used to store the number of times that the nodes are sampled.
\( N_\text{a} \) is the set of non-assigned root nodes of graph \( G \) which is initially set to \( V \).

Begin Algorithm
\( k \leftarrow 1; \)
While \( (k < K_{\text{max}}) \) Do
Inactive all learning automata;
\( t \leftarrow 1; \quad T_t \leftarrow 0; \)
Repeat
Select node \( v_i \) from non-assigned root nodes \( N_\text{a} \) randomly as root node;
\( N_\text{a} \leftarrow N_\text{a} \setminus v_i \); Set \( A_i \) to \( A_i \);
While \( (|L_\text{s}[v_i]| < |V| - 1 \text{ AND} a \text{ learning automaton can be activated}) \) Do
Learning automaton \( A_i \) is activated and then chooses an action according to its action probability vector;
Let the action chosen by \( A_i \) be edge \( e_{ij} \);
Visit and take a sample from chosen edge \( e_{ij} \);
\( \tau_i \leftarrow \tau_i \cup \{e_{ij}\} \);
\( \bar{w}_t \leftarrow \bar{w}_t + W_{ij} \);
\( L_\text{s}[v_i] \leftarrow L_\text{s}[v_i] + 1 \);
Disable action \( a_{ij} \) from action set of each inactive learning automaton connected to \( A_i \) to avoid the loop;
Set \( A_i \) to \( A_i \);
End While
\( T_t \leftarrow (T_{\text{sl}} + \bar{w}_t)/t_t \);
If \( (\bar{w}_t \leq T_{\text{sl}}) \) Then
Reward the chosen actions by all the activated learning automata along spanning tree \( \tau_t \); Else
Penalize the chosen actions by all the activated learning automata along spanning tree \( \tau_t \);
End If
If \( (\bar{w}_t \leq \bar{w}_{\text{off}}) \) Then
\( \bar{\tau} \leftarrow v_i \);
End If
\( P_C(t) = \prod_{v_i \in \omega}(\max(p(v_i))); \)
\( t \leftarrow t + 1 \);
Enable all the disabled actions;
Until \( (P_C(t) \geq P_{\text{max}} \text{ OR } t \geq T_{\text{max}}) \)
\( k \leftarrow k + 1 \);
End While
Sort \( L_\text{s} \) in descending order;
Construct an induced sub-graph \( G'\equiv(V', E', W') \) using \( \varphi |V| \text{ mostly visited nodes}; \)
End Algorithm

Fig. 4. The pseudo-code of the spanning tree sampling algorithm for stochastic graphs (SST-SG).

\( A_i \), which is assigned to node \( v_i \), is the set of outgoing edges of node \( v_i \) on the input graph. Each learning automaton in eDLA initially chooses its actions with equal probabilities. In eDLA, at any time, an LA can be in one of four levels: passive, active, fire and off. All learning automata are initially set at the passive level.

The eDLAS-SG consists of a number of stages. In each stage, a subgraph within the graph is traversed and the average weights of the edges along this subgraph are updated. To do this, one of the nodes in eDLA is randomly chosen by the firing function to be the starting node, while its activity level is set to active by the governing rule. Then, learning automaton \( A_i \) is fired (its activity level changes to fire and the activity level of its passive neighboring nodes changes to active) and chooses one of its actions, which corresponds to one of the edges of the fired node; at the same time, the level of the fired LA \( A_i \) changes to off due to the governing rule. Let the chosen action (edge) be \( e_{ij} \). Edge \( e_{ij} \) is sampled (visited) and the average weight of the sampled edge is recomputed. Next, one of the LAs in eDLA, whose activity level is active, is chosen and fired (its activity level changes to fire and the activity level of its passive neighboring nodes changes to active). The fired LA chooses one of its actions, which corresponds to an edge of the fired node, after which the activity level of the fired LA changes to off due to the governing rule. The process of selecting a learning automaton from the set of learning automata, whose activity level is active, in eDLA by the firing function, firing it, choosing an action by the fired LA, changing the activity levels of passive neighboring LAs to active LAs, computing the average weight of the chosen edge, and changing the activity level of fired LAs from fire to off as a result of the governing rule, is repeated until either the number of LAs whose activity level is off is reaches \( |V| \times \varphi \) or the set of LAs whose activity level is active is empty.

Once the traversal of subgraph \( \tau_t \) at stage \( t \) is terminated, the probability vectors of fired learning automata along the traversed path \( \tau_t \) are updated as follows. If the weight of \( t_i (\bar{w}_t) \)
Algorithm 3. DLAS-SG(G, ϕ, Kmax, Tmin)

Input: Stochastic graph G=(V, E, W), Sampling rate ϕ, Thresholds Kmax, Tmin.
Output: Sampled graph G’=(V’, E’, W’).
Initialization
Construct a DLA by assigning an automaton Ai to each node vi and initialize their action probabilities.
t denotes the iteration number of algorithm which is initially set to 1.
N denotes the average weight of all samples taken from edges along the path of πi.
Ls is an one-dimensional array with size |P| which is used to store the number of times that the nodes are sampled.
NS is the set of nodes which are not selected yet as starting nodes, this set is initially set to V.

Begin algorithm
Deactivate all learning automata;
t ← 1; T0 ← 0;
While (t < Kmax OR |Tt−1| ≥ Tmin) Do
πt ← {};
Select starting node vt, randomly from non-visited nodes NS;
NS ← NS \ {vt};
While (number of visited nodes ≤ ϕ|V| AND a learning automaton can be activated) Do
Learning automaton Ai is activated and then chooses an action according to its action probability vector;
Let the chosen action by Ai be edge eij;
Visit and take a sample from the chosen edge eij;
πt ← πt \ {eij} ;
πt ← πt U {eij} ;
πt ← πt \ {eij} ;
πt ← πt U {eij} ;
Ls[vij] ← Ls[vij] + 1;
Set Ai to Aj;
End While
Tt ← [(t−1)Tt−1 + Nt]/t;
If (Nt ≥ Tt) Then // favorable path
Reward the actions chosen by all the activated learning automata along path πt;
Else
Penalize the actions chosen by all the activated learning automata along path πt;
End If
(t ← t − 1;
End While
Sort Ls in descending order;
Construct an induced sub-graph G’=(V’, E’, W’) using ϕ|V| mostly visited nodes;
End Algorithm

Fig. 5. Pseudo-code of distributed learning automata based sampling algorithm for stochastic graphs (DLAS-SG).

is equal or greater than dynamic threshold Tt, computed as Tt = [(t−1)Tt−1 + Nt]/t, then the probability of actions chosen by all the fired learning automata along the traversed path τt is increased, or decreased otherwise, according to the learning algorithm. Before a new stage begins, the activity levels of all learning automata in eDLA are changed and the activity level of all learning automata in eDLA are set to passive. A stage begins if the maximum number of stages (Kmax) has not been reached and the difference between two dynamic thresholds in two consecutive stages is equal or greater than a predefined threshold Tmin.

When the execution of stages is over, the nodes visited during each stage are sorted in descending order according to the number of subgraphs along which a node has appeared. The sampled network is then constructed by considering an induced subgraph of the input network, whose node set contains a given number of mostly visited nodes, while its edge weights are the average weights of edges estimated during the execution of the algorithms. The pseudo-code of eDLAS-SG as Algorithm 4 for stochastic graphs is given in Fig. 6.

5. Simulation results

In this section, the performance of the proposed sampling algorithms for stochastic graphs is studied on several well-known real and synthetic stochastic networks. Table 1 describes the characteristics of the test networks used for the experiments. The real networks are: HT09 [54], Facebook-like-OPSAHL-UC SOCIAL [55], Cit-HepTh [56], Facebook-wall [57], and LKML-Reply [58]. The synthetic networks are: ER-SG is generated based on the Erdős–Rényi model [59] which is utilized widely in the literature as a random network; WS-SG is made based on the Watts–Strogatz model [5] which is a synthetic small-world network; and BA-SG is created based on Barabási–Albert model [6] which is a synthetic scale-free network. The nodes for all synthetic networks are: N = 10,000 and p = 0.15 for ER-SG; p = 0.2 for WS-SG, and m0 = m = 5 for BA-SG. The edge weights of the real networks are random variables, which represent the number of activities among individuals during a specified timestamp for each network. The edge weights of synthetic networks are random variables with Weibull distribution, whose parameters are a = 0.32, b = 0.17. The chosen parameters are adopted from an empirical observation from Twitter concerning the distribution of lifetime tweets [60].

5.1. Evaluation measures

Since the main objective of network sampling is to approximate the entire input network in order to estimate network measures or select a representative subgraph structure, the approach taken to evaluate network sampling algorithms can often involve comparing network measures of the original and sampled structural networks. Therefore, the distance between network measure values (e.g., clustering coefficient) and network measure distributions
Algorithm 4. eDLAS-SG(G, φ, K_max, T_max)

Input: Stochastic graph G=(V, E, W), Sampling rate φ, Thresholds K_max, T_max.

Output: Sampled graph G*(V*, E*, W*).

Initialization

Construct an eDLA by assigning an automaton A_i to each node v_i.

Initialize action probabilities of automata and set the activity level of each LA to Passive.

I is the iteration number of algorithm which is initially set to 1.

Pa is the set of learning automata with activity level of Passive which is initially set to {v_1, v_2, ..., v_n}.

Ac is the set of learning automata with activity level of Active which is initially set to empty.

Of is the set of learning automata with activity level of Off which is initially set to empty.

Fi is the learning automaton with activity level of Fire which is initially set to empty.

w_j is the average weight of all samples taken from edges along subgraph of τ_j.

Ls is a one dimensional array with size |V| which is used to store the number of times that the nodes are sampled.

N(v_j) is a function that returns the all adjacent nodes of v_j with Passive level.

Begin algorithm

I ← 1;

While (I ≤ K_max OR |T_i−T_s|≥T_max) Do

 Pa ← {v_1, v_2, ..., v_n};

 Fi ← {};

 Ac ← {};

 Of ← {};

 Select a starting node v_i randomly by firing function and change its activity level of A_i to Active level;

 Ac ← Ac ∪ {A_i};

 Pa ← Pa \ A_i;

 τ_j ← τ_j ∪ {v_j};

 Ls[v_j] ← Ls[v_j] + 1;

 While (|Of| ≤ φ|V| AND |Ac|>0) Do

 Select one Active LA A_j by firing function and then chooses an action according to its action probability vector; 

 Let the action chosen by A_j be edge e_j;

 Take a sample from the chosen edge e_j;

 τ_j ← τ_j ∪ {e_j};

 w_j ← w_j + w_j;

 Ls[v_j] ← Ls[v_j] + 1;

 Fi ← A_j;

 Ac ← N(v_j) \ v_j;

 Pa ← Pa \ Ac;

 Of ← Of \ |Fi|;

 Fi ← {};

 End While

 T_s ← ((I−1)T_s + w_j)/I;

 If (w_j ≥ T_s) Then

 Reward the actions chosen by all the fired learning automata with activity level Of in subgraph τ_j;

 Else

 Penalize the actions chosen by all the fired learning automata with activity level Of in subgraph τ_j;

 End If

 I ← I + 1;

 End While

 Sort list of visited node Ls in descending order;

 Construct an induced sub-graph G*(V*, E*, W*) using φ|V| mostly visited nodes;

 End Algorithm

Fig. 6. Pseudo-code of extended distributed learning automata based sampling algorithm for stochastic graphs (eDLAS-SG).

(e.g., strength distribution) of the original networks and those of the sampled network is calculated in order to evaluate the quality of the sampled network. The interested readers may refer to [61] for alternative approaches to network sampling algorithms. In this paper, since we focus on using network measure value and distributions to study network properties, we use the Kolmogorov-Smirnov (KS) D-statistics, the Kendall’s rank correlation coefficient and relative error (RE) as distance measures for evaluation as well as relative cost in order to compare the performance of different sampling algorithms proposed in this paper for stochastic graphs. These evaluation measures, which function as evaluating criteria, are described below.

5.1.1. Kolmogorov-Smirnov D-statistics (KS-D)

Kolmogorov-Smirnov D-statistics (KS-D) [23] represent one of the statistical test methods that is commonly used for assessment the distance between two cumulative distribution functions (CDFs). The KS-D approach computes the maximum vertical distance between the cumulative distribution function of the original distribution from the original graph and that of estimated distribution from the sampled graph. This measure is defined as:

$$D(P, Q) = \max_x \{|P(x) - Q(x)|\}$$  (12)

where P and Q are two CDFs of the original and estimated data, respectively, while x represents the range of the random variable. The KS-D approach is sensitive to both locations and shapes of distributions, as well as being an appropriate measure for the similarity of the distribution. As the KS-D value between the original network and the sampled network gets closer to zero, this means that both networks have a greater similarity, while the two networks have a greater difference the closer the value is to the unit.

5.1.2. Relative error (RE)

Relative error (RE) can be applied to assess the accuracy of the results for a single parameter, which is defined by the following equation:

$$RE = \frac{|P - Q|}{P}$$  (13)

where P and Q denote the values of real and sampled parameters (i.e., real clustering coefficient and estimated clustering coefficient) in the original data and obtained samples data, respectively [17].

5.1.3. Kendall’s rank correlation coefficient

The central nodes of the graph are specified using network centralities. In the experimentation, we use Kendall’s τ rank correlation coefficient [62] to investigate how much the ranking list of
central nodes between the original network and the sampled network is preserved. Kendall’s τ is computed as follows:

$$\tau = \frac{P_c - P_d}{N_T}$$

(14)

where $P_c$ is the set of concordant pairs, $P_d$ is the set of discordant pairs and $N_T$ is the number of pairs in total.

5.1.4. Relative cost (RC)

Relative cost (RC), which can be applied to assess the cost of a sampling algorithm can be defined as the cost regarding the total number of times that the edges in the graph are traversed (i.e., taking samples from the edges) during the traversal of the graph ($C_c$), plus the cost regarding the total number of operations performed during post-processing ($C_p$), divided by the total number of edges in the graph ($E$), as given by the following equation:

$$RC = \frac{C_c + C_p}{E}$$

(15)

where constant $c_1$ is the coefficient cost of traversing edges (taking samples) and constant $c_2$ is the coefficient cost of performing an operation during the post-processing phase for assigning rank to nodes.

5.2. Experimental settings

The control parameters for the proposed algorithms are set as follows. For all algorithms, the maximum number of iterations $K_{\text{max}}$ is set as $n \times \psi$, where $n$ is the number of nodes for each instance graph. For all algorithms, the reinforcement scheme used for updating the action probability vector of learning automata is $L_{R,1}$, with $\alpha = 0.05$. For both SPS-SG and SST-SG algorithms, the maximum number of stage-number $T_{\text{max}}$ is $10 \times n$, while the threshold for the product of probability $P_{\text{max}}$ is set at 0.9. For DLAS-SG and eDLAS-SG threshold $T_{\text{min}}$ is set at 0.05. The reported results are the averages taken over 30 runs.

### Table 1

<table>
<thead>
<tr>
<th>Network</th>
<th>Nodes</th>
<th>Edges</th>
<th>Type</th>
<th>Directed</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HT09 [54]</td>
<td>113</td>
<td>2196</td>
<td>Face-to-face conversation</td>
<td>N</td>
<td>ACM Hypertext 2009 dynamic contact network, Torino, IT, over three days. Network of sent messages between the users of an online community of students from the University of California, Irvine.</td>
</tr>
<tr>
<td>Facebook-like-OPSAH-UCSOCIAL [55]</td>
<td>1899</td>
<td>59,835</td>
<td>User-user messaging</td>
<td>Y</td>
<td>Collaboration graph of authors of scientific papers from the arXiv's High Energy Physics-Theory (Hep-Th) section.</td>
</tr>
<tr>
<td>Cit-HepTh [56]</td>
<td>27,770</td>
<td>352,807</td>
<td>Author–author collaborations</td>
<td>Y</td>
<td>Network of wall posts from the Facebook New Orleans networks. Networks of the communication network of the Linux kernel mailing list.</td>
</tr>
<tr>
<td>Facebook-wall [57]</td>
<td>63,731</td>
<td>1,545,684</td>
<td>User-user wall post</td>
<td>Y</td>
<td>Synthetic random graph based on the Erdős-Rényi model, the edge weights are Weibull random variables, with $a = 0.32$, $b = 0.17$.</td>
</tr>
<tr>
<td>LKML-Reply [58]</td>
<td>63,399</td>
<td>1,096,440</td>
<td>User-user reply</td>
<td>Y</td>
<td>Synthetic scale-free random graph based on Barabási-Albert model graph, the edge weights are Weibull random variables, with $a = 0.32$, $b = 0.17$.</td>
</tr>
<tr>
<td>ER-SG</td>
<td>10,000</td>
<td>8,060,206</td>
<td>Synthetic ER stochastic graph</td>
<td>N</td>
<td>Synthetic world random graph generated using the Watts-Strogatz model, the edge weights are Weibull random variables, with $a = 0.32$, $b = 0.17$.</td>
</tr>
<tr>
<td>WS-SG</td>
<td>10,000</td>
<td>10,001,633</td>
<td>Synthetic WS stochastic graph</td>
<td>N</td>
<td>Synthetic scale-free random graph generated using the Barabási-Albert model graph, the edge weights are Weibull random variables, with $a = 0.32$, $b = 0.17$.</td>
</tr>
<tr>
<td>BA-SG</td>
<td>10,000</td>
<td>58,749</td>
<td>Synthetic BA stochastic graph</td>
<td>N</td>
<td>Synthetic scale-free random graph based on Barabási-Albert model graph, the edge weights are Weibull random variables, with $a = 0.32$, $b = 0.17$.</td>
</tr>
</tbody>
</table>

Table 2: Average ranking of Friedman’s test for the comparing sampling algorithms.

<table>
<thead>
<tr>
<th>Test results</th>
<th>SPS-SG</th>
<th>SST-SG</th>
<th>DLAS-SG</th>
<th>eDLAS-SG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average ranking</td>
<td>2.96</td>
<td>3.48</td>
<td>2.21</td>
<td>1.33</td>
</tr>
<tr>
<td>Ranking</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

5.3. Experimental results

To investigate the performance of the proposed sampling algorithms for stochastic graphs, a number of experiments are conducted on several stochastic real and synthetic networks, as described in Table 1. Note that the proposed sampling algorithms for stochastic graphs are abbreviated as follows: shortest path sampling (SPS) for stochastic graphs as SPS-SG, spanning tree sampling (SST) for stochastic graphs as SST-SG, distributed learning automata (DLAS) for stochastic graphs as DLAS-SG, and extended distributed learning automata (eDLAS) for stochastic graphs as eDLAS-SG.

5.3.1. Experiment I

This experiment is carried out to study the performance of the proposed sampling algorithms for stochastic graphs (SPS-SG, SST-SG, DLAS-SG, eDLAS-SG) with respect to stochastic strength and stochastic clustering coefficient, as defined in Section 3. In this paper, the stochastic version of strength distribution and the clustering coefficient is used for this experiment. These measures as key measures have been widely studied by many researchers in order to understand the connectivity and transitivity of the network, respectively. In this experiment, the sampling rate varied from 10% to 30% in increments 5%. Meanwhile, the results for all algorithms and for each test networks, are given in terms of KS-D for strength distribution in Fig. 7, KS-D for clustering coefficient distribution in Fig. 8 and RE for the average clustering coefficient in Fig. 9. According to the results, we may conclude that, for all the test networks, the performance of the sampling algorithms in terms of the
Fig. 7. Comparing sampling algorithms in terms of KS-D for strength distribution.
Fig. 8. Comparing sampling algorithms in terms of KS-D for clustering coefficient distribution.
Table 3
p-values of the post hoc comparisons with $\alpha = 0.05$ for the comparing sampling algorithms (SPS-SG, SST-SG, DLAS-SG, eDLAS-SG).

<table>
<thead>
<tr>
<th>i</th>
<th>Sampling algorithms</th>
<th>$z = (R_{i} - R_{j})/SE$</th>
<th>$p$-value</th>
<th>Holm $p$-value</th>
<th>Shaffer $p$-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>DLAS-SG vs. eDLAS-SG</td>
<td>12.0535</td>
<td>0.0001</td>
<td>0.0083</td>
<td>0.0083</td>
</tr>
<tr>
<td>5</td>
<td>SPS-SG vs. DLAS-SG</td>
<td>9.1671</td>
<td>0.0100</td>
<td>0.0100</td>
<td>0.0167</td>
</tr>
<tr>
<td>4</td>
<td>SST-SG vs. eDLAS-SG</td>
<td>7.0824</td>
<td>0.0125</td>
<td>0.0125</td>
<td>0.0167</td>
</tr>
<tr>
<td>3</td>
<td>SPS-SG vs. SST-SG</td>
<td>4.9771</td>
<td>0.0001</td>
<td>0.0167</td>
<td>0.0167</td>
</tr>
<tr>
<td>2</td>
<td>SST-SG vs. DLAS-SG</td>
<td>4.9060</td>
<td>0.0002</td>
<td>0.0250</td>
<td>0.0250</td>
</tr>
<tr>
<td>1</td>
<td>SPS-SG vs. eDLAS-SG</td>
<td>2.8864</td>
<td>0.0038</td>
<td>0.0500</td>
<td>0.0500</td>
</tr>
</tbody>
</table>

Note: Nemenyi’s procedure rejects those hypotheses that have an unadjusted $p$-value $\leq 0.0083$. Holm’s procedure rejects those hypotheses that have an unadjusted $p$-value $\leq 0.0083$. Shaffer’s procedure rejects those hypotheses that have an unadjusted $p$-value $\leq 0.0083$.

Table 4
Kendall’s $\tau$ rank correlation between two ordered central lists of original and sampled networks using node strength.

<table>
<thead>
<tr>
<th>Sampling rate (%)</th>
<th>SPS-SG</th>
<th>SST-SG</th>
<th>DLAS-SG</th>
<th>eDLAS-SG</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\tau$</td>
<td>Rank</td>
<td>$\tau$</td>
<td>Rank</td>
</tr>
<tr>
<td>10</td>
<td>0.24</td>
<td>4</td>
<td>0.33</td>
<td>3</td>
</tr>
<tr>
<td>15</td>
<td>0.24</td>
<td>4</td>
<td>0.37</td>
<td>3</td>
</tr>
<tr>
<td>20</td>
<td>0.24</td>
<td>4</td>
<td>0.46</td>
<td>3</td>
</tr>
<tr>
<td>25</td>
<td>0.25</td>
<td>4</td>
<td>0.55</td>
<td>3</td>
</tr>
<tr>
<td>30</td>
<td>0.25</td>
<td>4</td>
<td>0.55</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 5
Kendall’s $\tau$ rank correlation between two ordered central list of original and sampled networks using node closeness.

<table>
<thead>
<tr>
<th>Sampling rate (%)</th>
<th>SPS-SG</th>
<th>SST-SG</th>
<th>DLAS-SG</th>
<th>eDLAS-SG</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\tau$</td>
<td>Rank</td>
<td>$\tau$</td>
<td>Rank</td>
</tr>
<tr>
<td>10</td>
<td>0.04</td>
<td>2</td>
<td>0.34</td>
<td>4</td>
</tr>
<tr>
<td>15</td>
<td>0.54</td>
<td>2</td>
<td>0.38</td>
<td>4</td>
</tr>
<tr>
<td>20</td>
<td>0.55</td>
<td>2</td>
<td>0.38</td>
<td>4</td>
</tr>
<tr>
<td>25</td>
<td>0.55</td>
<td>3</td>
<td>0.41</td>
<td>4</td>
</tr>
<tr>
<td>30</td>
<td>0.57</td>
<td>3</td>
<td>0.43</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 6
Kendall’s $\tau$ rank correlation between two ordered central list of original and sampled networks using node betweenness.

<table>
<thead>
<tr>
<th>Sampling rate (%)</th>
<th>SPS-SG</th>
<th>SST-SG</th>
<th>DLAS-SG</th>
<th>eDLAS-SG</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\tau$</td>
<td>Rank</td>
<td>$\tau$</td>
<td>Rank</td>
</tr>
<tr>
<td>10</td>
<td>0.63</td>
<td>1</td>
<td>0.37</td>
<td>4</td>
</tr>
<tr>
<td>15</td>
<td>0.69</td>
<td>1</td>
<td>0.39</td>
<td>4</td>
</tr>
<tr>
<td>20</td>
<td>0.70</td>
<td>1</td>
<td>0.47</td>
<td>4</td>
</tr>
<tr>
<td>25</td>
<td>0.72</td>
<td>1</td>
<td>0.48</td>
<td>4</td>
</tr>
<tr>
<td>30</td>
<td>0.79</td>
<td>1</td>
<td>0.59</td>
<td>4</td>
</tr>
</tbody>
</table>

5.3.2. Experiment II

This experiment was conducted to compare the proposed algorithms with respect to their computational complexity (cost). For this experiment, the sampling rate varied from 15% to 30% with a 5% interval. The comparison was performed with respect to the relative cost of sampling algorithms, as defined by Eq. (15) for $c_1 = 1$, $c_2 = 1$. The results of this experiment for different sampling algorithms are presented in Fig. 10. From the results shown in Fig. 7, we observe that, for all sampling algorithms, relative cost increased as the sampling rate increased. The higher cost for a higher sampling rate results from additional samples and post-processing on the network as performed by the algorithms. The results also indicate that, for the same sampling rate, eDLAS-SG and DLAS-SG required lower costs than other sampling algorithms. Among the sampling algorithms, SST-SG has the highest cost because of the computation of the spanning tree with all nodes for each iteration.

5.3.3. Experiment III

This experiment is sought to examine how much the ranking of central nodes based on network centrality measures between the original and the sampled networks is preserved. For this purpose, by using the proposed network measures, as defined in Section 3 for stochastic graphs, we ranked the nodes of the original and sampled networks with the proposed sampling algorithms presented in Section 4, after which the correlation coefficient between two ordered lists was computed via Kendall’s $\tau$ rank correlation coefficient defined by Eq. (14), for different sampling rates. The average results of this experiment are given in Table 4 for node.

The aforementioned measures increased as the sampling rate increased. From the results shown in Fig. 7, it is clear that, in terms of KS-D for strength distribution, eDLAS-SG outperformed other sampling algorithms for at least seven out of eight test networks. According to the results shown in Fig. 8, in terms of KS-D for clustering coefficient distribution, eDLAS-SG outperformed other sampling algorithms for at least four out of eight test networks, while SPS-SG outperformed other sampling algorithms for at least three out of eight test networks. From the results shown in Fig. 9, in terms of RE for the average clustering coefficient, eDLAS-SG outperformed other sampling algorithms for at least three out of eight test networks.

Moreover, in order to examine whether a statistical significant difference exists among the proposed sampling algorithms, we conducted a series of multi-comparison statistical tests (Friedman and Iman-Davenport) with a significance interval of 95% ($\alpha = 0.05$) [63]. The Friedman test [64] is used as a nonparametric test to rank the proposed sampling algorithms with respect to all the aforementioned evaluation measures, with the best algorithm being assigned the rank of 1, the second best, rank 2 etc. As a statistical analysis, Friedman’s test was 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, Holm and Shaffer tests were conducted as post hoc methods (if significant differences were detected).

The rankings obtained by Friedman’s test are given in Table 2. Since the $p$-value computed by the Friedman’s test is 7.67E-11 which is below the significance interval of 95% ($\alpha = 0.05$), thus, a significant difference can be said to exist among the observed results. According to the results of average rankings, based on statistical significance in Table 2, one can rank eDLAS-SG and SST-SG as first and last, respectively. Post hoc methods (Holm and Shaffer tests) were also performed to obtain the $p$-values, which are given in Table 3. In this test, the null hypothesis was that all the sampling algorithms were equivalent; if the null-hypothesis was rejected, we were able to compare all the weighted sampling algorithms with each other using the Nemenyi’s test [65]. The $p$-values in Table 3 also indicates that eDLAS-SG outperforms the other sampling algorithms (SPS-SG, SST-SG, DLAS-SG) with a significance interval of 95% ($\alpha = 0.05$).
strength. Table 5 for node closeness and Table 6 for node betweenness. Table 4 shows that DLAS-SG with a sampling rate of 10–20% and eDLAS-SG with a sampling rate 25–30% have a higher Kendall’s τ values than SPS-SG and SST-SG using node strength. This result indicates that DLAS-SG and eDLAS-SG tend to samples nodes with high strength. From the results given in Table 5, one may conclude that eDLAS-SG has the highest Kendall’s τ values using node closeness among all sampling algorithms. This also shows that eDLAS-SG has a tendency to take samples from nodes with high closeness. As shown in Table 6, SPS-SG involves the Kendall’s τ values using
Fig. 10. Comparing the proposed sampling algorithms in terms of relative cost for varying sampling rate.
Acknowledgments

This research was in part supported by a grant from IPM. (No. CS1395-4-67).

References


