A Streaming Sampling Algorithm for Social Activity Networks using Fixed Structure Learning Automata

Mina Ghavipour [a], Mohammad Reza Meybodi [b]

[a,b] *Department of Computer Engineering and IT, Amirkabir University of Technology, Tehran, Iran*

Abstract: Social activity networks are formed from activities among users (such as wall posts, tweets, emails, and etc.), where any activity between two users results in an addition of an edge to the network graph. These networks are streaming and include massive volume of edges. A streaming graph is considered to be a stream of edges that continuously evolves over time. This paper proposes a sampling algorithm for social activity networks, implemented in a streaming fashion. The proposed algorithm utilizes a set of fixed structure learning automata. Each node of the original activity graph is equipped with a learning automaton which decides whether its corresponding node should be added to the sample set or not. The proposed algorithm is compared with the best streaming sampling algorithm reported so far in terms of Kolmogorov-Smirnov (KS) test and normalized L_1 and L_2 distances over real-world activity networks and synthetic networks presented as a sequence of edges. The experimental results show the superiority of the proposed algorithm.

Keywords: Social networks, Activity networks, Network sampling, Streaming sampling, Learning automata

1. INTRODUCTION

Recently, social network has become another means of communication integrated into the regular patterns of social life which is evident from the popularity of social networks such as Facebook, Twitter and LinkedIn. Thus, analyzing these networks is increasingly important for discovering patterns of interactions among individuals, and investigating structural attributes and evolution of the networks over time, as well as developing algorithms that operate on these networks. Despite their popularities, the dynamicity and large size of most social networks make it difficult or impossible to study the entire network. Therefore, it is often necessary to sample smaller subgraphs from the original network to be used for analyzing the larger network. There are many properties to characterize a graph, such as degree distribution, clustering coefficient, shortest path length, and etc. Sampling algorithms must sample representative subgraphs that have similar properties as compared to original graphs. Leskovec and Faloutsos [1] first defined representative subgraphs sampling. They proposed several algorithms to verify the goodness of sampling algorithms, and developed a set of empirical rules for scaling up the measurements of the sample, to get estimates for the original graph. Ebbes et al. [2] evaluated the performance of nine different sampling techniques in recovering the underlying structural characteristics of social networks. They focused on four characteristics of interest for marketers, including degree, clustering coefficient, betweenness centrality, and closeness centrality. Work in [3, 4] studied the statistical properties of the samples taken from scale-free networks by three sampling algorithms: node, edge and random walk sampling.

Most of the works on sampling from social networks [5–10] have assumed the network graph to have moderate size and static structure, and focused only on producing samples that match graph properties well. In other words, these algorithms assume that the graph can fit in the main memory and the access to the entire graph at any step is possible. However, these assumptions are not realistic for many real world networks. For instance, consider social activity networks formed from communications among users (such as

wall posts, tweets, emails, and etc.), where any activity between two users results in an addition of an edge to the network graph. These networks are streaming and include the massive volume of edges. A streaming graph is considered to be a stream of edges that continuously evolves over time and is clearly too large to fit in the memory [11]. In the domain of streaming graphs, traditional sampling algorithms are no longer appropriate. When the original network has too many edges to fit in the main memory, sampling can only be done sequentially (one edge at a time), since random accesses on disk incur large I/O costs. In the static domain, topology based sampling methods require the random exploration of a node's neighbors (which requires many passes over the edge stream if only sequential accesses are allowed). Node sampling method also requires to randomly access the entire node set of the network graph. Thus, none of these methods would be appropriate for sampling such a large scale network. In addition, in some cases it is necessary to analyze a dynamic network over time for many applications, such as for investigating how the structure of communities evolve over time, and discovering patterns of interactions among individuals. In these cases also, static graph sampling algorithms are not appropriate as they have no ability to update the sampled subgraph using edges that occur over time. Therefore, several snapshots at different points in time must be taken from the original network and for each snapshot, the sampling process has to be done entirely again to obtain an updated sample of that time point. As a result, it is necessary to develop sampling algorithms that can address the complexities of streaming domain.
While there are many research works on graph streams [12–18], however only few research works [11, 19–21] have focused on sampling representative subgraphs from the streaming graphs. Work in [19] has attempted to maintain cascaded summaries of the graph stream by uniformly sampling a subset of edges from the stream. Aggarwal et al. [21] have designed a reservoir sampling method for structural summarization. Authors in [20] have developed a time-based streaming sampling method for activity graphs and then tested it on dynamic real-world networks. In [11], several algorithms which are the adaptation of static sampling algorithms to streaming sampling have been presented. All of these streaming sampling algorithms run in a single pass over the stream and take into consideration both the stream evolution and the massive size of the networks. It has been reported that the algorithm called PIES has the highest performance. The algorithm PIES has been tested on several real-world datasets and its superiority has been shown over other algorithms. To the best of our knowledge, PIES is the best streaming sampling method reported so far.

1.1 The Motivation

The algorithm PIES has two drawbacks: (1) streaming edges are each sampled independently. Thus, connectivity and clustering of the original graph is less likely to be preserved in sampled subgraphs. Using partial graph induction (sampling some of the edges incident on the sampled nodes) can help the algorithm to recover only some of the original connectivity, (2) the incident nodes of any sampled edge replace nodes randomly selected from the sample. Therefore, it is possible that nodes with high activity are replaced while there exist some low activity or even isolated nodes in the sample. As a result of these drawbacks, PIES performs well for sparse, less clustered graphs and the performance of the algorithm decreases as the graph becomes denser and more clustered. This is while online social networks tend to exhibit high levels of clustering [22, 23] and it also has been observed that the density of these network increases over time [24, 25]. Considering these in mind, our contribution is to develop a streaming sampling algorithm based on learning automata which, while maintaining the advantages of PIES (such as running in a single pass over the stream and considering the stream evolution), can overcome these drawbacks and produce sample subgraphs with high quality also for dense and high clustered graphs.

1.2 Our Contribution

Our contributions in this paper are as follows:

- We propose a learning automata based streaming sampling algorithm, called FLAS, which runs in a single pass over the edge stream and maintains a dynamic sample while the original graph is streaming. Each node of the original activity graph is equipped with a learning automaton which decides whether its corresponding node should to be added to the sample subgraph or not. Using learning automata help our proposed algorithm FLAS to overcome the drawbacks of the algorithm PIES [11] (as the best streaming sampling method reported so far) and produce more connected sample subgraphs.

- We conduct extensive experiments on real-world activity networks and synthetic networks. It is shown that our sampling algorithm FLAS is superior in terms of the representativeness of sampled subgraphs, and competitive in terms of space complexity as compared to the PIES algorithm. The representativeness ability of the proposed algorithm is tested in terms of Kolmogorov-Smirnov (KS) test for degree, clustering coefficient, k–core and path length distributions and also in terms of normalized L_1 and L_2 distances respectively for eigenvalues and network values.
- According to the experimental results, FLAS obtains an improvement of about 66% for degree distribution, of about 69% for clustering coefficient distribution, of about 64% for k–core distribution, of about 48% for path length distribution, of about 78% for eigenvalues, and of about 76% for network values at the cost of time complexity as compared to PIES. For an input graph $G(V, E)$ (where V is the set of nodes and E is the set of edges), FLAS needs $O(|E|\log|V|)$ access to disk, while the number of disk accesses needed by PIES is $O(|E|)$. It is worth mentioning that if there are enough main memory space both algorithms FLAS and PIES have the same time complexity of $O(|E|)$.

The rest of the paper is organized as follows. In the next section, we present a background and related work on sampling algorithms. The learning automata are introduced in Section 3. We then describe our learning automata-based algorithm for sampling from activity networks and compare the proposed algorithm with the best streaming sampling algorithm reported so far in Sections 4 and 5, respectively. Section 6 is the conclusion.

2. BACKGROUND AND RELATED WORK

In this section, we formalize the problem of sampling from real-world networks and discuss graph sampling algorithms in literature. These algorithms can be grouped into two classes namely, static graph sampling algorithms and streaming graph sampling algorithms.

2.1 Problem Definition

In this paper, we consider the graph sampling problem as follows. Let $G(V, E)$ be an undirected graph with the set of nodes $V = \{v_1, v_2, \ldots, v_n\}$ and the edges' set $E = \{e_1, e_2, \ldots, e_m\}$ that is presented as a stream of edges arranged in an arbitrary sequential order. Given a sampling fraction f, the task is to sample a representative subgraph $G_s(V_s, E_s)$ of size $n_s = f\text{n}$ (i.e. $n_s = |V_s| = f \times |V|$), that will match the properties of the original graph G.

2.2 Static Graph Sampling

The assumption of a static graph has been made by the traditional sampling algorithms which can be conceptually categorized into three groups: algorithms based on randomly selecting nodes, randomly selecting edges, and topology based sampling algorithms.

In node sampling (NS), nodes are chosen uniformly at random from the original graph. Then the selected nodes along with all the edges among them constitute the sampled subgraph. Authors in [26] showed that uniform node sampling does not accurately retain power-law degree distribution. Despite NS includes all edges for a sampled node set, the work in [3] showed that it is less likely to preserve the original level of connectivity. Leskovec and Faloutsos [1] proposed another variations of the node sampling in which the probability of a node being selected is proportional to either its degree or PageRank weight. They showed that simple uniform NS performs well in many situations.

Similarly to the node sampling, one can also select edges uniformly at random, referred to as edge sampling (ES). In this situation, the sampled subgraph is constructed by including the sampled edges as well as both incident nodes of each chosen edge. ES is slightly biased towards high degree nodes, since there are more edges incident to them. Due to the independent sampling of edges, the original connectivity is less likely to be preserved in this sampling method [3].

While there are many sampling algorithms based on topology, the common idea in all these algorithms is to first select a node uniformly at random and then explore the neighboring nodes of the chosen node. Examples are Breath-First Search (BFS) [27–29], Snowball Sampling (SBS) [30], Forest Fire Sampling (FFS) [13, 24], Random Walk Sampling (RWS) [4, 31]. Work in [32] has shown that BFS is biased towards high degree nodes, but this bias can be corrected by analytical solutions suggested in [8]. Authors in [1] observed that the FFS method matches well the properties of the original graph and produces accurate samples. FFS behaves almost the same as BFS sampling, the difference is that in FFS only a fraction of neighbors is followed at each round. Lu and Li [33] compared the RWS method with NS and ES, and shown that RWS performs much better than two other sampling algorithms on Twitter dataset. Different variations of RWS have been developed in literature such as weighted random walk (WRW) [34], re-weighted random walk (RWRW) [35], metropolis-hastings random walk (MHRW) [36, 37], m-dependent random walk [38], random walks with jumps [39], and random walk based node sampling (ICLA-NS) [5].

2.3 Streaming Graph Sampling

The traditional sampling algorithms assume that graphs are static and fit in the main memory, and their goal is to produce samples that preserve graph properties well. However, in the case of an activity network, since users interact repeatedly over time, the network graph is dynamic and the stream of edges is considered to be continuously time varying and include too many edges. Under these conditions, the traditional sampling techniques are not appropriate, since they consider the graph at one point in time and thus the samples taken by them may become less relevant over time because of evolution. In addition, these techniques incur large I/O costs when the graph is too large to fit into the memory.

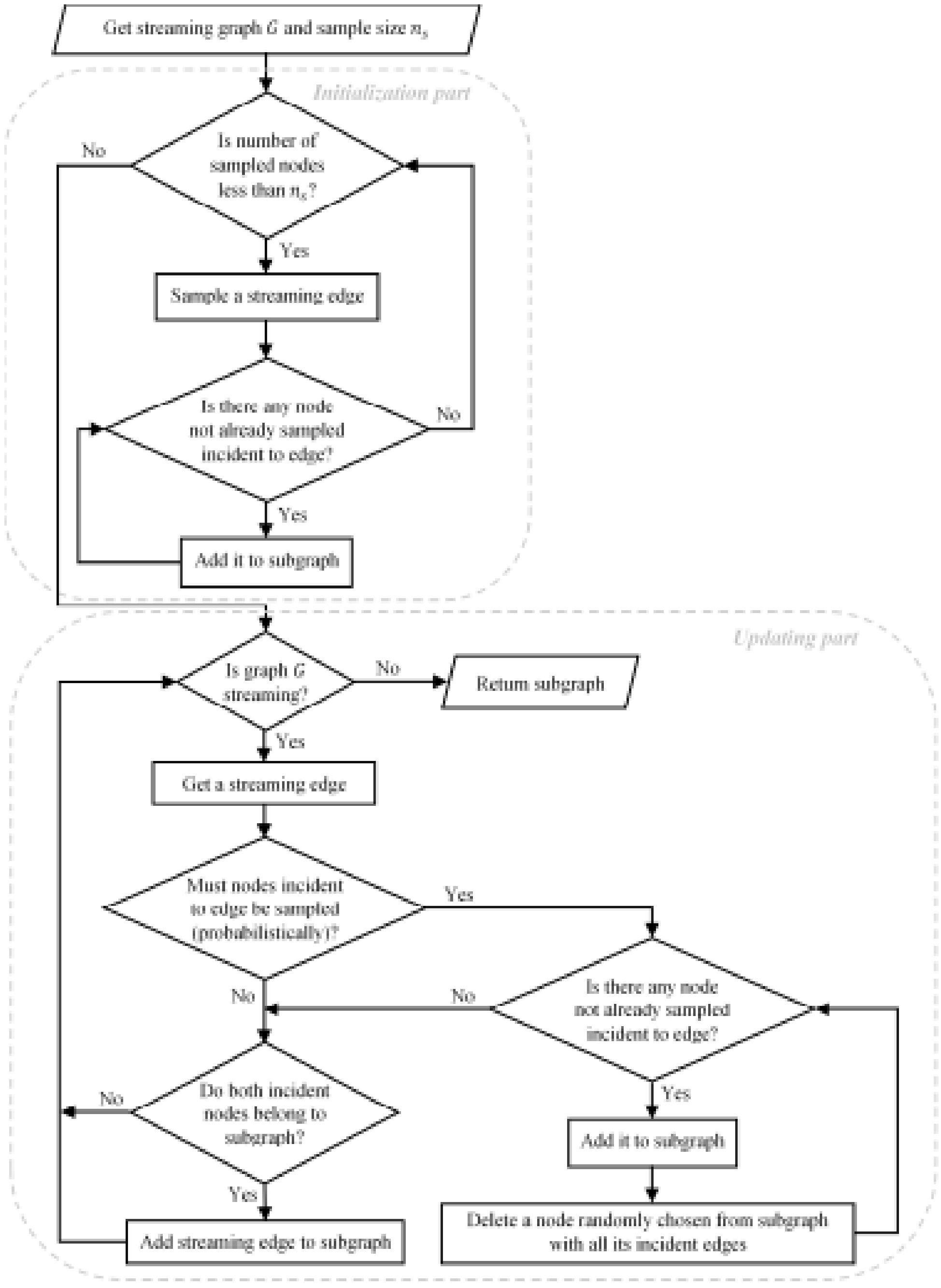
In the real world, there are numerous examples of activity networks such as Facebook wall, Twitter posts, and Email communications. Due to the importance of studying such dynamic large scale networks, researchers have developed techniques for sampling representative subgraphs from these streaming graphs. Authors in [19] utilized a min-wise hash function to sample almost uniformly from the set of all edges which has been at any time in the graph stream. The sampled edges were used later to maintain cascaded summaries of the stream. Work in [21] proposed a reservoir sampling method based on min-wise hash sampling of edges in order to maintain structural summaries of the underlying graph. These structural summaries are designed to create dynamic and efficient models for detecting outlier in graph streams. Ahmed et al. [20] developed a time-based sampling technique for sampling from activity graphs presented as a sequence of edges ordered over time. This method randomly selects a timestamp on the activity timeline of the graph and samples the nodes incident on edges that have occurred in a time window starting from that timestamp. The algorithm repeats this process in a streaming fashion until the required fraction of nodes are collected. Finally, the sample set includes the selected nodes and any future edges that involve these nodes. Authors have compared their method with traditional sampling algorithms such as node sampling and Forest Fire sampling. Work in [11] have dealt with graph sampling problem by outlining a spectrum of computational models for sampling algorithms, ranging from the simplest model based on assumption of static graphs to the more challenging model of sampling from graph streams. They proposed several sampling algorithms based on the concept of graph induction generalized across the spectrum from static to streaming. In static domain, they proposed a sampling algorithm called induced edge sampling (ES-i) which was a combination of edge-based node sampling method and the graph induction. The authors proved the better performance of their proposed algorithm ES-i by comparing it with traditional sampling algorithms. In streaming domain, Ahmed et al. [11] addressed the massive size of edges (that is too large to fit in memory) and continuously evolving edge stream over time, and adapted static sampling algorithms for streaming graphs. They presented streaming variations of node, edge and topology-based sampling, as well as a streaming variation of the algorithm ES-i, referred to as partially-induced edge sampling (PIES), which all run in a single pass over the stream of edges. As reported in [11], PIES preserves more accurately the underlying properties of tested datasets compared to other streaming algorithms. This algorithm is biased to high degree nodes and provides a dynamic sample while the original graph is streaming. The pseudo code and schematic diagram for PIES are depicted respectively in Figures Fig. 1 and Fig. 2. The algorithm receives as input a streaming graph presented as an arbitrarily ordered sequence of edges and adds the first m edges incident to n_s nodes to the sample set. Then, it scans the rest of the stream and randomly samples edges such that any streaming edge is sampled with probability $\gamma = \frac{m}{t}$ if at least one of two nodes incident to that edge does not belong to the sample set, and otherwise that edge is sampled with probability $\gamma = 1$. For any sampled edge, the incident nodes replace former sampled nodes chosen uniformly at random. As noted in [11], PIES achieves better result for graphs that are sparse and less clustered. To the best of our knowledge, the algorithm PIES which is compared with the sampling algorithm proposed in this paper is the best streaming sampling method reported so far.

```
ALGORITHM PIES. Partially-induced edge sampling
Input: Original graph $G(V, E)$ , Sample size $n_s$
Output: Sampled subgraph $G_s(V_s, E_s)$
Initialization part
    $V_s = \emptyset$ ; $E_s = \emptyset$;
    $t = 1$;
    while $|V_s| < n_s$ do
        $(v_i, v_j) = e_t$;
        $E_s = E_s \cup \{e_t\}$;
        if $v_i \notin V_s$ then  $V_s = V_s \cup \{v_i\}$ ;
        if $v_j \notin V_s$ then  $V_s = V_s \cup \{v_j\}$ ;
        $t = t + 1$;
    end
Updating part
    $m = |E_s|$;
    while graph $G$ is streaming do
        $(v_i, v_j) = e_t$;
        $\gamma = \frac{m}{t}$;
        Generate p at random in interval (0,1);
        if $p \le \gamma$ then
            for each node $v_k$ incident to $e_t$ do
                if $v_k \notin V_s$ then
                    $V_s = V_s \cup \{v_k\}$;
                    Delete node $v_{k'}$ randomly chosen from $V_s$ with all its incident edges;
                end
            end
        end
        if $v_i \in V_s$ and $v_j \in V_s$ then  $E_s = E_s \cup \{e_t\}$;
        $t = t + 1$;
    end
```
Fig. 1. The pseudo code of PIES algorithm.

3. LEARNING AUTOMATA

Learning automata (LA) [40, 41] is a reinforcement learning approach that its learning process can be described as follows. The automaton has a finite set of actions and it is restricted to select one of them at each step. The chosen action serves as the input to the environment with which the automaton interacts, and the environment responds with a reinforcement signal with a certain probability. Based on these interactions, the learning automaton learns the optimal action, which is the action with the minimum penalty probability.
The random environment can be defined mathematically by a triple $\langle \alpha, \beta, c \rangle$, where $\alpha = \{\alpha_1, \alpha_2, \dots, \alpha_r\}$ is a finite input set, $\beta = \{\beta_1, \beta_2, \dots, \beta_k\}$ denotes a set of values taken by the reinforcement signal (an output set), and $c = \{c_1, c_2, \dots, c_r\}$ represents a set of penalty probabilities, where each element c_i corresponds to one input action α_i. The random environment is called a non-stationary environment, if the penalty probabilities vary with time, and it is said to be stationary, if the penalty probabilities are constant. The interaction between a learning automaton and its random environment is shown in Fig. 3.
Learning automata has found many applications, such as graph sampling [5, 10], fuzzy membership function optimization [42], vertex coloring [43], to mention a few. LA can be categorized into two main families: a

learning automaton is called a variable structure learning automaton (VSLA) if its transition function and output function vary over time, otherwise it is called a fixed structure learning automaton (FSLA). A FSLA is a quintuple $\langle \alpha, \beta, \Phi, \mathcal{F}, \mathcal{G} \rangle$, where:



Get streaming graph G and sample size n_s
Initialization part
Is number of sampled nodes less than n_s?
No
Yes
Sample a streaming edge
Is there any node not already sampled incident to edge?
No
Yes
Add it to subgraph
Updating part
Is graph G streaming?
No
Return subgraph
Yes
Get a streaming edge
Must nodes incident to edge be sampled (probabilistically)?
Yes
No
Is there any node not already sampled incident to edge?
No
Yes
Add it to subgraph
Delete a node randomly chosen from subgraph with all its incident edges
Do both incident nodes belong to subgraph?
No
Yes
Add streaming edge to subgraph

Fig. 2. The schematic diagram of PIES algorithm.

1) $\alpha = \{\alpha_1, \alpha_2, \dots, \alpha_r\}$ is the set of the actions that automaton chooses from.
2) $\beta = \{0,1\}$ is automaton's set of inputs where it receives a penalty, if $\beta = 1$ and receives a reward otherwise.
3) $\Phi = \{\phi_1, \phi_2, \dots, \phi_{rN}\}$ indicates its set of states, where N is called the depth of memory of the automaton.

4) $\mathcal{F}: \Phi \times \beta \rightarrow \Phi$ illustrates the transition of the state of the automaton on receiving an input from the environment. $\mathcal{F}$ can be stochastic.
5) $\mathcal{G}: \Phi \rightarrow \alpha$ is the output function of the automaton. The action taken by the automaton is determined according to its current state. This means that the automaton selects action α_i if it is in any of the states $\{\phi_{(i-1)N+1}, \phi_{(i-1)N+2}, \dots, \phi_{iN}\}$. The state $\phi_{(i-1)N+1}$ is considered to be the most internal state, and ϕ_{iN} is considered to be the boundary state of action α_i, indicating that the automaton has the most and the least certainty in performing the action α_i, respectively.

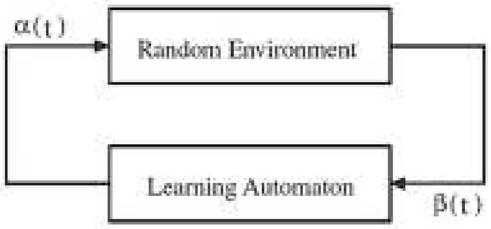


Fig. 3. The relationship between a learning automaton and the random environment

The action chosen by the automaton is applied to the environment which in turn emits a reinforcement signal β. On the basis of the received signal β, the state of the automaton is updated and the new action is chosen according to the functions $\mathcal{F}$ and $\mathcal{G}$ respectively. There exist different types of FSLA based on the state transition function $\mathcal{F}$ and the output function $\mathcal{G}$. $L_{2N,2}$, $G_{2N,2}$ and Krinsky are the FSLA types used in this paper. We will introduce these automata in the following subsections.
The learning ability of a learning automaton is investigated by comparison with a pure-chance automaton. If there is no prior information available, there is also no basis on which different actions can be distinguished. In such a case, the automaton chooses its available actions at random, i.e. by pure chance. The pure-chance automaton is used as the standard for comparison, such that any automaton, that is said to learn, is expected to do at least better than a pure-chance automaton. The comparison can be done in terms of the average penalty $M(t)$ received by an automaton. $M(t)$ for a pure-chance automaton is a constant value and is denoted by $M(0)$. For a learning automaton to do better than a pure chance automaton, the expected value of its average penalty $\mathrm{E}[M(t)]$ must be less than $M(0)$ at least asymptotically as $t \rightarrow \infty$.

Definition 1. A learning automaton is said to be expedient if

$$\lim_{t\to\infty} E[M(t)] < M(0)$$

Expediency means that the average penalty received by an automaton decreases over time as a result of the increase of learning [41].

3.1 The Tsetlin Automaton ($L_{2N,2}$)

The Tsetlin automaton [44] is an extension of the more simple $L_{2,2}$ automaton. This automaton has $2N$ states, that are denoted by $\phi_1, \phi_2, \dots, \phi_{2N}$, and two actions α_1 and α_2. These states keep track of the automaton's prior behavior and its received feedback. The automaton chooses an action based on the current state it resides in. That is, action $\alpha_1$is chosen by the automaton if it is in the states $\phi_1, \phi_2, \dots, \phi_N$, while if the current state is $\phi_{N+1}, \phi_{N+2}, \dots, \phi_{2N}$ action α_2 is chosen. The automaton moves towards its most internal state whenever it receives a reward, and conversely, on receiving a penalty, it moves towards its boundary state or to the boundary state of the other action. For instance, consider that the automaton is in state ϕ_N. The automaton moves to state ϕ_{N-1} if it has been rewarded and it moves to state ϕ_{2N} in the case of punishment. However, if the current state is either ϕ_1 or ϕ_{N+1} , the automaton remains in the mentioned states until it keeps on being rewarded. The state transition graph of $L_{2N,2}$ automaton is depicted in Fig. 4.
As mentioned before, the state transition function $\mathcal{F}$ can be considered as a stochastic function [44]. In such a case, on receiving a signal from the environment, the transition of the automaton among its states is not deterministic. For instance, when an action results in a reward, the automaton may move one state towards

the boundary state with probability $\gamma_1 \in [0,1)$, and move one state towards its most internal state with probability $1-\gamma_1$, and reverse this procedure using probabilities γ_2 and $1-\gamma_2$ when the response of the environment is penalty. In some situations where rewarding a favorable action may be preferable more than penalizing an unfavorable action, one can set $\gamma_1 = 0$ and $\gamma_2 \in [0,1)$. These settings result in state transitions of $L_{2N,2}$ automaton become deterministic when the automaton receives a reward ($\beta = 0$), as shown in Fig. 4. But, in the case of punishment the $L_{2N,2}$ automaton will transit among its states stochastically as shown in Fig. 5. By considering $\gamma_1 = 0$, the automaton will be expedient for all values of γ_2 in interval $[0,1)$ [41].

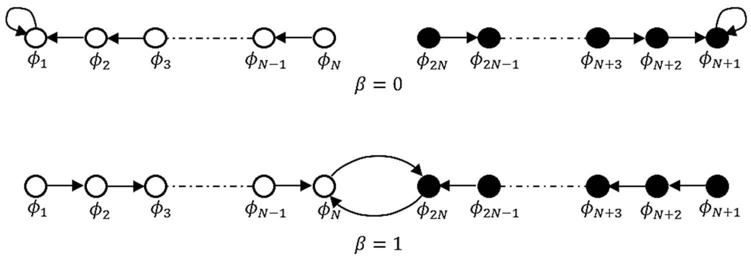


Fig. 4. The state transition graph for $L_{2N,2}$ automaton

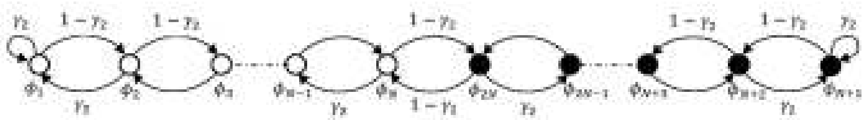


Fig. 5. The state transition graph for $L_{2N,2}$ in case of punishment

3.2 The TsetlinG Automaton ($G_{2N,2}$)

The TsetlinG automaton [44] is another type of the FSLA used in this paper. This automaton behaves exactly the same as the $L_{2N,2}$ automaton, except for the times when it is being punished where the automaton moves from state ϕ_N to ϕ_{N+1} and from state ϕ_{2N} to state ϕ_1. The state transition graph of $G_{2N,2}$ automaton is illustrated in Fig. 6. When the state transition function of $G_{2N,2}$ automaton is stochastic and $\gamma_1 = 0$ and $\gamma_2 \in [0,1)$, the automaton transits among its states on receiving a reward as shown in Fig. 6 and in the case of punishment as shown in Fig. 7.

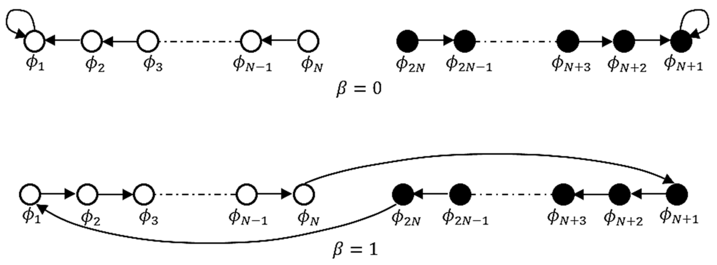


Fig. 6. The state transition graph for $G_{2N,2}$ automaton

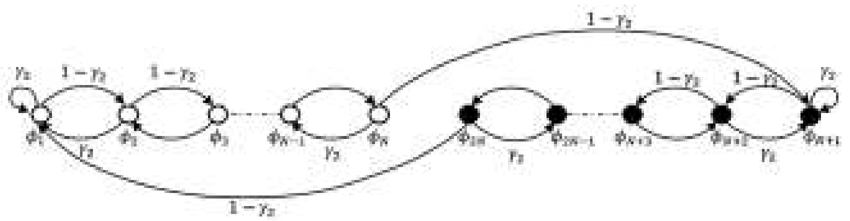


Fig. 7. The state transition graph for $G_{2N,2}$ in case of punishment

3.3 The Krinsky Automaton

In this subsection we briefly describe another type of the fixed structure learning automata, namely the Krinsky automaton [44]. When the chosen action by this automaton results in a penalty it acts exactly like the $L_{2N,2}$ automaton. However, in situations where the automaton is rewarded, any of the states $\phi_1, \phi_2, \dots, \phi_N$ pass to the state ϕ_1 and any of the states $\phi_{N+1}, \phi_{N+2}, \dots, \phi_{2N}$ transit to the state ϕ_{N+1}. Therefore, N successive penalties are needed for the automaton to switch from its current action to the other one. One can note the state transition graph of the introduced automaton in Fig. 8. In the case of a stochastic state transition function and for settings $\gamma_1 = 0$ and $\gamma_2 \in [0,1)$, the automaton behaves the same as before deterministically when it is rewarded (see Fig. 8) and on receiving a penalty, the behaviour of Krinsky automaton is identical to the $L_{2N,2}$ automaton as shown in Fig. 5.

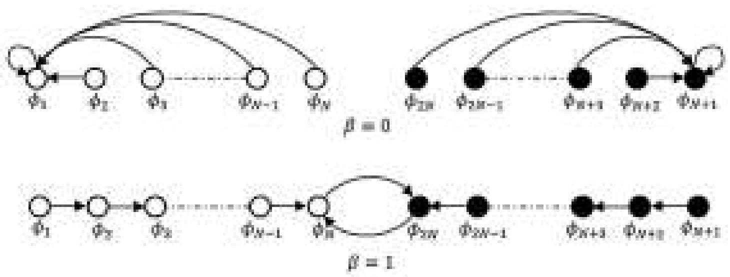


Fig. 8. The state transition graph for Krinsky automaton

4. THE PROPOSED STREAMING SAMPLING ALGORITHM

In this section, a fixed structure learning automata-based sampling algorithm, called FLAS is presented for sampling from activity graphs. The inputs are an activity graph $G\ (V, E)$ presented as a sequence of edges in an arbitrary order, as well as the sample size n_s. The output of the algorithm is a representative sample $G_s(V_s, E_s)$ ($|V_s| = n_s$) that will match the properties of the graph G. Here, we describe the algorithm FLAS on the basis of the state transitions of the $L_{2N,2}$ automaton with the stochastic transition function as depicted in Figures Fig. **4** and Fig. **5**. The $L_{2N,2}$ automaton behaves deterministically when it is rewarded and stochastically when it receives a penalty, i.e. $\gamma_1 = 0$ and $\gamma_2 \in [0,1)$ (for more details, see Section 3). Since γ_1 is set to zero, in the rest of the paper we ignore it and refer to γ_2 as γ.

The pseudo code and schematic diagram of the sampling algorithm FLAS are given respectively in Figures Fig. **9** and Fig. **10**. FLAS consists of two parts, initialization part and updating part.

Initialization part. At first, an initial sample graph is created by successively adding the edges one by one from the beginning of the stream to E_s and their incident nodes to V_s until the required number of nodes is sampled ($|V_s| = n_s$). When a node v_k is visited for the first time, it is equipped with a fixed structure learning automaton LA_k with two actions α_1 and α_2. Initially, the states of learning automata corresponding to the nodes in V_s are set to the boundary state of action α_2 ($\forall v_k \in V_s$: the current state of LA_k is set to ϕ_{2N}, i.e. $\Phi^k = \phi_{2N}$).

Remark 1. Note that for an automaton being in one of the states of action α_2 indicates that the corresponding node is in V_s and being in one of the states of action α_1 indicates that the corresponding node is in $V - V_s$. The state of the action of learning automaton indicates the strength of membership of the corresponding node.

Updating part. In this part, the algorithm consecutively processes the remaining edges of the stream and keeps on updating the sample as long as the graph is streaming. The decision whether to sample a streaming edge or not is taken based on the states of the learning automata corresponding to the nodes incident on that edge. When a node is visited for the first time, it is equipped with a fixed structure learning automaton residing in the boundary state of action α_1 (i.e. ϕ_N).

Considering this in mind, each time the algorithm is confronted with an edge (an activity) $e_t = (v_i, v_j)$, one of the following cases may be encountered.

```
ALGORITHM FLAS. FLA-based sampling
Input: Original graph $G(V,E)$ , Sample size $n_s$
Output: Sampled subgraph $G_s(V_s,E_s)$
Initialization part
    $V_s = \emptyset$ ; $E_s = \emptyset$;
    $t = 1$;
    while $|V_s| < n_s$ do
        $(v_i, v_j) = e_t$;
        $E_s = E_s \cup \{e_t\}$;
        for each node $v_k$ incident to $e_t$ do
            if $v_k \notin V_s$ then
                $V_s = V_s \cup \{v_k\}$;
                Assign an $L_{2N,2}$ automaton $LA_k$ to $v_k$; $\Phi^k = \phi_{2N}$;
            end
        end
        $t = t + 1$;
    end
Updating part
    while graph $G$ is streaming do
        $(v_i, v_j) = e_t$;
        for each node $v_k$ incident to $e_t$ do
            if $v_k \in V_s$ then          \\ Reward automaton $LA_k$
                if $\Phi^k \ mod\ N \neq 1$ then $\Phi^k = \Phi^k - 1$;
            else                           \\ Penalize automaton $LA_k$
                if $v_k$ is visited for first time then Assign an $L_{2N,2}$ automaton $LA_k$ to $v_k$; $\Phi^k = \phi_N$;
                Generate p at random in interval (0,1);
                if $p \leq \gamma$ then
                    if $\Phi^k \ mod\ N \neq 1$ then $\Phi^k = \Phi^k - 1$;
                else
                    if $\Phi^k \ mod\ N \neq 0$ then
                        $\Phi^k = \Phi^k + 1$;
                    else
                        $V_s = V_s \cup \{v_k\}$;
                        Choose sampled node $v_{k'}$ whose corresponding automaton's state is closest to its boundary state;
                        Delete $v_{k'}$ with all its incident edges from $G_s$;
                        $\Phi^k = \phi_{2N}$; $\Phi^{k'} = \phi_N$;
                    end
                end
            end
        end
        if $v_i \in V_s$ and $v_j \in V_s$ then $E_s = E_s \cup \{e_t\}$;
        $t = t + 1$;
    end
```

Fig. 9. The pseudo code of FLAS algorithm when L_{2N-2} automaton is used

1) Both automata LA_i and LA_j are in one of the states of action α_2.
2) One of the automata is in one of the states of action α_1 and the other automaton is in one of the states of action α_2.
3) Both automata LA_i and LA_j are in one of the states of action α_1.

The actions taken by the algorithm for each of the above cases are described in the next three paragraphs.

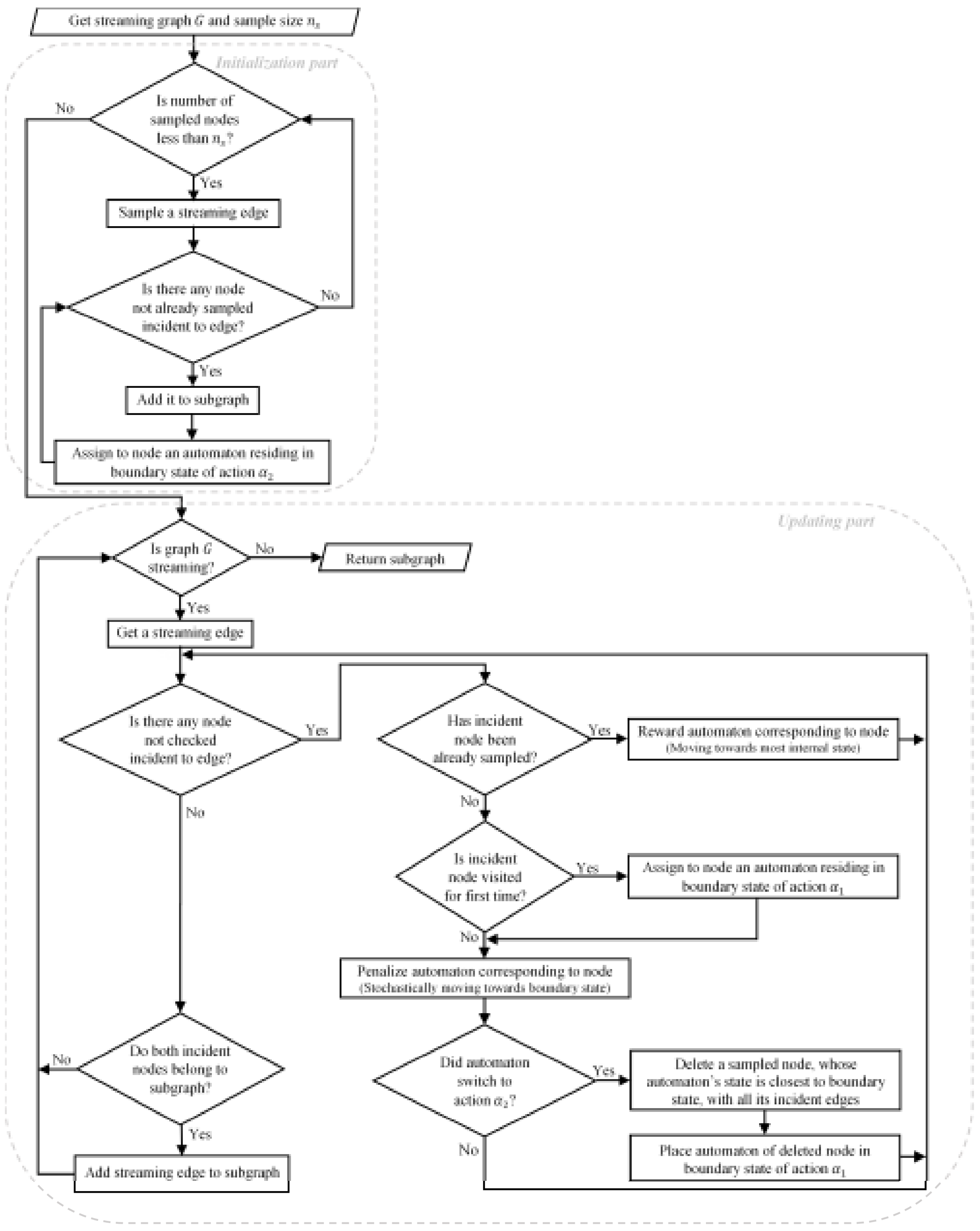

Get streaming graph G and sample size n_s
Initialization part
Is number of sampled nodes less than n_s?
No
Yes
Sample a streaming edge
Is there any node not already sampled incident to edge?
No
Yes
Add it to subgraph
Assign to node an automaton residing in boundary state of action α_2
Updating part
Is graph G streaming?
No
Return subgraph
Yes
Get a streaming edge
Is there any node not checked incident to edge?
Yes
No
Has incident node been already sampled?
Yes
Reward automaton corresponding to node (Moving towards most internal state)
No
Is incident node visited for first time?
Yes
Assign to node an automaton residing in boundary state of action α_1
No
Penalize automaton corresponding to node (Stochastically moving towards boundary state)
Did automaton switch to action α_2?
Yes
Delete a sampled node, whose automaton's state is closest to boundary state, with all its incident edges
Place automaton of deleted node in boundary state of action α_1
No
Do both incident nodes belong to subgraph?
No
Yes
Add streaming edge to subgraph

Fig. 10. The schematic diagram of FLAS algorithm when L_{2N-2} automaton is used

If LA_i and LA_j are both in one of the states of action α_2, both automata are rewarded (i.e. $\Phi^k = \Phi^k - 1 : k \in \{i, j\}$) and then the edge e_t is added to E_s .
If one of the automata (for example LA_i) is in one of the state of action α_1 and the other automaton (LA_j) is in one of the states of action α_2 then the automaton LA_i receives penalty (moves one state towards the most internal state with probability γ and towards the boundary state with probability $1 - \gamma$) and the automaton LA_j is rewarded (i.e. $\Phi^j = \Phi^j - 1$). If as a result of penalizing LA_i, LA_i switches to the action α_2, then the edge e_t is added to E_s. In order to preserve the constraint $|V_s| = n_s$, a node from V_s whose corresponding automaton's state is closest to its boundary state along with its incident edges will be removed. If there exist more than one such a node, one of them, say $v_{i'}$, is selected at random. The state of the automaton of the removed node $v_{i'}$ changes to the boundary state of the action α_1 (i.e. $\Phi^{i'} = \phi_N$).
If both LA_i and LA_j are in one of the states of action α_1 then they are penalized. If after the penalization both automata switch to the action α_2 then the incident nodes on edge e_t are added to V_s and the edge e_t is added to E_s. In order to preserve the constraint $|V_s| = n_s$, two nodes from V_s whose corresponding automata's states are closest to their boundary states along with all the edges incident to those nodes will be removed. If there exist more than two nodes with this property, two of them are selected at random. The state of each of the learning automata of the removed nodes then changes to its boundary state of the action α_1.

Remark 2. By using fixed structure learning automata (FSLA) in our proposed algorithm in comparison with variable structure learning automata (VSLA), we obtain some advantages. The first advantage is that using FSLA reduces the computational cost of our algorithm. FSLA keeps an account of the number of rewards and penalties received for each action. It continues performing whatever action it was earlier using as long as it is rewarded, and switches from one action to another only when the number of penalties exceeds the depth of memory N. However, VSLA chooses its action randomly based on the action probability vector kept over the action set. Therefore, it is probable that the automaton switches to another action even although it has received reward for doing the current action, until the selection probability of an action converges to one. As a result, the sample subgraph will incur many changes and the complexity of our algorithm will increase (for more details, see subsection 4.2). As the second advantage, different behavioral variations of an observer can be modeled by assigning a FSLA to each node of the graph. Each FSLA plays the role of an observer which keeps track of the history of the activities of its node with other nodes in order to decide whether it is time to be included or omitted from the sample. Depending on the type of FSLA (for more details, see Section 3), different observers with different behaviors can be implemented. For example, when $L_{2N,2}$ is used we expect a conservative behavior from the observer whereas when $G_{2N,2}$ is used we expect an optimistic behavior from the observer. This possibility of modelling different behavioral variations does not exist when using VSLA.

4.1 Discussion

Similar to the algorithm PIES developed in [11], the proposed algorithm FLAS scans the edge stream in a single pass, has a selection bias to high activity nodes, uses the concept of graph induction and retains a dynamic sample while the graph G is streaming. The main difference between FLAS and the algorithm PIES is in the way that edges are sampled and their incident nodes are added to the sample set. Since the algorithm PIES is based on edge sampling concepts, it samples any streaming edge independently at random. Therefore, PIES is less likely to preserve connectivity and clustering of the original graph, and using partial graph induction (namely sampling some of the edges incident on the sampled nodes in the forward direction) help PIES to recover only some of the connectivity losses. In addition, PIES adds the incident nodes of any sampled edge to the sample by replacing the nodes which are selected uniformly at random from V_s. Therefore, it is possible that, while there exist some low activity or even isolated nodes (isolated nodes are a result of removing all the incident edges to replaced nodes) in the sample, the nodes with high degree in V_s are replaced. Because of these drawbacks, namely independent sampling of edges and the random replacement of sampled nodes, PIES performs well for sparse, less clustered graphs and the performance of the algorithm decreases as the graph becomes denser and more clustered.

While maintaining the advantages of PIES, our proposed algorithm FLAS attempts to overcome these two drawbacks by using fixed structure learning automata. In FLAS, addition of an edge to the sample and then selection of nodes of V_S to be replaced by the nodes incident on that edge are done on the basis of the decisions made by the learning automata. In fact, the learning automaton of each node somehow keeps track of the history of the activities of its node with other nodes. Once an edge is encountered, the learning automata of the nodes which are incident on that edge collectively decide whether the edge and (or) the incident nodes to be added to the sample or not. Therefore, the probability of being sampled of an edge depends on other edges which have been already observed from the edge's incident nodes, and any edge incident on the nodes with high activity has higher probability to be sampled than the edges incident on the nodes with low activity. In addition, for a sampled edge the learning automata corresponding to the nodes in V_S determine which sampled nodes must be replaced by the incident nodes to the edge. This way, the lower the activity of a sampled node, the higher the probability that it is selected to be replaced. As a result, using fixed structure learning automata, high activity nodes have the higher probability of inclusion in subgraphs sampled by FLAS as comparing to PIES (since PIES is slightly biased towards nodes with high activity and it is more likely to lose some of the sampled high activity nodes because of the random replacement).
It is well-known that many real world networks are scale-free in which the degree distribution follows power-law, where $P(k) \sim k^{-\gamma}$, and most of them have $2 < \gamma < 3$ [45]. It also has been shown in [46] that scale-free networks are extremely vulnerable to removing high degree nodes (which represent hubs in these networks), breaking into many isolated fragments. Based on this, authors in [4] proposed that for a scale-free network with $\gamma \lesssim 3$ the weighted sampling, where the weight of each node is proportional to its degree, can produce subgraphs which share the same topological properties with the original graph. Especially, they shown that by sampling high degree nodes the sampled subgraph will have the degree distribution nearly same as the original one. Clearly, all sampling methods cause a downward bias in the degree distribution of sampled subgraph. The reason is that these methods always select a subset of the nodes or edges of the original graph in order to construct the sampled graph. Sampling high degree nodes overestimates the original degree distribution (results in an upward bias), considering the degrees of the sampled nodes as they are observed in the original graph. As also shown in [11], a combination of high degree bias as well as the graph induction can offset the downward bias of the sampled degree distribution. Considering these in mind, the proposed algorithm FLAS attempts to sample nodes with high degree and produce more connected subgraphs which preserve better the properties of the original graph [1].

4.2 Complexity analysis

In this section, we analyze the space and time complexity of the proposed algorithm FLAS.

Space complexity. The input graph G and the learning automata of the nodes not belonging to the sample graph will be always kept on the disk. In the memory, we keep the sampled edges and the learning automata corresponding to the sampled nodes. Each learning automaton requires only one space of memory for its state value. Hence the total memory space needed by the algorithm is $O(|E_S| + |V_S|)$.

Time complexity. The analysis given below is a worst case analysis. The worst case occurs if in all iterations of the updating part of the algorithm every time an edge is taken from the stream of the edges to be processed, the nodes incident on the edge are not in the set of sampled nodes and also their corresponding learning automata change their actions.
We assume that the algorithm uses an unsorted list data structure to store the input graph and the sampled edge stream. We also assume that for storing the learning automata on both disk and memory the algorithm uses a B-Tree data structure [47]. Table 1 shows the time complexity of operations such as insert, delete and search on these data structures. We also assume that each disk access takes a certain amount of time referred to as c. Considering these in mind, the time complexity of our algorithm in the worst case is computed as follows.
In the initialization part, the algorithm adds each edge read from disk to the sampled edge stream in $O(1)$ time. For each incident nodes of the sampled edge, FLAS inserts its corresponding automaton into B-Tree

stored on memory if it does not exist which takes $O(\log|V_S|)$ time. Therefore, the time complexity of the initialization part is $O(|E_S|\log|V_S| + c|E_S|)$.

Table 1. Time complexity of operations

Dataset	Insert	Delete	Search
Unsorted list	O(1)	O(*n*)	O(*n*)
B-Tree	O(log *n*)	O(log *n*)	O(log *n*)

In the updating part, every time a streaming edge is read from disk, the algorithm first searches B-Tree in memory for the learning automata of its incident nodes which takes $O(\log|V_S|)$ time. Since only the automata corresponding to the sampled nodes are kept in memory, FLAS then has to search the B-Tree stored on disk in $O(c\log(|V| - |V_S|))$ time. After that, the found automata are moved to the memory, penalized and then inserted in B-Tree stored on memory in $O(c + \log|V_S|)$ time. In order to preserve the constraint on the sample size, FLAS searches B-Tree in memory for two automata whose states are closest to the boundary state and moves these two automata to disk which takes $O(|V_S| + c\log(|V| - |V_S|))$ time. Finally, all the edges incident to two replaced nodes are removed and the new streaming edge is added to the sampled edge stream that takes $O(|E_S|)$ time. Hence, the time taken by the updating part in the worst case is $O(c(|E| - |E_S|)\log(|V| - |V_S|) + |V_S| + |E_S|)$.
Considering the above analysis, the total time complexity of our algorithm FLAS will be $O(|E_S|\log|V_S| + c|E_S| + c(|E| - |E_S|)\log(|V| - |V_S|) + |V_S| + |E_S|)$. Since the term $|E|\log(|V| - |V_S|)$ has the largest growth rate in this formula and usually $|V_S| \ll |V|$, the time taken by FLAS in the worst case can be considered to be $O(|E|\log|V|)$.

5. EXPERIMENTAL EVALUATION

In this section, we assess the efficiency of the proposed algorithm FLAS on several real world networks. We utilize social networks from Flickr [48] and Facebook from New Orleans City [49], a collaboration network from CondMAT and a citation network from ArXiv HepPH [50]. Table 2 summarizes the global statistics of these real world networks.

Table 2. Characteristics of used datasets

Dataset	Nodes	Edges	Density	Avg. Path	Global Clustering
Flickr	820878	6625280	1.9E-5	6.5	0.116
Facebook	46952	183412	2E-4	5.6	0.085
HepPH	34546	420877	7E-4	4.33	0.146
CondMAT	23133	93439	4E-4	5.35	0.264

5.1 Network Statistics

The performance of a sampling algorithm is measured by determining how well the subgraphs sampled by it match properties of the original graph. The statistics that we consider in our experiments are: degree, clustering coefficient, k–core, and path length distributions, eigenvalues and network values. We present a formal definition of these properties below:

Degree Distribution. The degree of a node in graph G is the number of connections or edges the node has to other nodes. The degree distribution $P(d)$ is then considered to be the fraction of nodes in the network with degree $d > 0$. Thus if there are n_d nodes in the network with degree d, we have

$$P(d) = \frac{n_d}{|V|}$$

Clustering Coefficient Distribution. The clustering coefficient for a node is defined as the proportion of links between the nodes within its neighborhood to the number of links that could possibly exist between them. The clustering coefficient distribution $P(c)$ is then calculated as

$$P(c) = \frac{n_c}{|V'|}$$

where $0 \le c \le 1$, n_c illustrates the number of nodes in graph G with clustering coefficient c, and V' is the set of nodes with degree greater than 1.

K–core Distribution. The k–core of graph G is defined as largest subgraph of G in which all nodes have degree at least k. A node has coreness k if it belongs to the k–core but not to the $(k+1)$–core. The k–core Distribution $P(k)$ is computed as the fraction of nodes having a coreness $k \ge 0$,

$$P(k) = \frac{n_k}{|V|}$$

where n_k denotes the number of nodes in graph G with coreness k.

Path Length Distribution. The shortest path length denotes the fewest number of hops required to reach from a node to another node. The path length distribution $P(h)$ is considered to be the fraction of pairs of nodes in graph G with shortest path length $h > 0$,

$$P(h) = \frac{n_h}{|V|^2}$$

where n_h is the number of pairs with shortest path length h.

Eigenvalues. The eigenvalue λ of the adjacency matrix A of graph G is computed as

$$Av = \lambda v$$

where v is the eigenvector of A associated with the eigenvalue λ. Eigenvalues are known as the basis of spectral graph analysis [11]. In our experiments, we consider the largest 25 eigenvalues of a graph G.

Network Values. Network values denotes the distribution of the eigenvector components corresponding to the largest eigenvalue of the adjacency matrix A of graph G [11]. In the experiments, the largest 100 network values of a graph G are considered.

5.2 Evaluation Measures

To assess the distance between any statistic in the original graph and that in the sampled subgraph, we use the following distance measures in this paper:

Kolmogorov-Smirnov (KS) statistic. The KS statistic is commonly used for measuring the agreement between two cumulative distribution functions (CDF) [51]. This statistic can be computed as the maximum vertical distance between two distributions:

$$KS = \max_x \left| F(x) - \acute{F}(x) \right|$$

where x is the range of the random variables, F and $\acute{F}$ denote two CDFs, and $0 \le KS \le 1$. In this paper, we use the KS statistic for computing the distance between the true distribution of the original graph and the estimated distribution from the sampled subgraph for degree, clustering coefficient, k–core, and path length distributions.

Normalized L_1 distance. The normalized L_1 distance (i.e. normalized manhattan distance) computes the distance between two positive m-dimensional real vectors [11],

$$L_1 = \frac{1}{m} \sum_{i=1}^{m} \frac{|p_i - \acute{p}_i|}{p_i}$$

where p and $\acute{p}$ are respectively the true vector and the estimated vector. In our experiment, we use normalized L_1 distance to measure the distance between two vectors of eigenvalues from the original and the sampled graphs.

Normalized L_2 distance. The normalized L_2 distance (i.e. normalized euclidean distance) measures the distance between two vectors when the components of these vectors are fractions [11]. This distance measure is computed as

$$L_2 = \frac{\|p - \acute{p}\|}{\|p\|}$$

In this paper, the normalized L2 distance is utilized for computing the distance between two vectors of network values related to the original and the sampled graphs.

5.3 Experimental Results

To evaluate the performance of the proposed algorithm FLAS, two groups of experiments on the real world networks described in Table 2 are conducted. In the first group of experiments (Experiments I, II, and III), we investigate the impact of the various parameters such as the type of fixed structure learning automata and its parameters (γ and N) on the performance of the proposed algorithm in order to find their best values. In the second group of experiments (Experiments IV, V, VI, VII, VIII, and IX), the proposed algorithm considering the best setting for the parameters is compared with algorithm PIES. In the experiments, the sampling fraction f varies from 0.1 to 0.3 with increment 0.025. For each sampling fraction, we report the average results of 30 independent runs. In each run, we utilize a random permutation of the edge stream as the input to the algorithm to make the results independent of any stream ordering. It is ensured that the different variations of the proposed algorithm and the algorithm PIES use the same streaming orders.

EXPERIMENT I

In this experiment, we study the impact of the type of fixed structure learning automata used by the algorithm FLAS on the performance in terms of KS distance for degree, clustering coefficient and path length distributions. The parameter depth N is set to 6 and transition probability γ is set to 0.7 for all types of FSLA. In order to provide statistical confidence, we perform a parametric test (t-Test at the 95% confidence interval). The test results for the sampling fraction 0.2 are summarized in Tables 3, 4 and 5.

Table 3. The results of statistical test for different types of FSLA in terms of the KS distance for degree distribution

FSLA Types	Dataset	Test Result		
		Mean KS Distance ($KS_x - KS_y$)	Difference Significance	Performance
$x = L_{2N,2}$, $y = G_{2N,2}$	Flickr	0.0006	1.6135E-5	×
	Facebook	0.0010	3.0266E-7	×
	HepPH	0.0289	9.5271E-32	×
	CondMAT	0.0041	1.0114E-9	×
$x = L_{2N,2}$, $y =$ Krinsky	Flickr	-0.0075	4.5900E-11	√
	Facebook	-0.0111	4.7440E-16	√
	HepPH	-0.0253	1.0676E-30	√
	CondMAT	-0.0202	6.9965E-17	√
$x = G_{2N,2}$, $y =$ Krinsky	Flickr	-0.0091	5.0764E-14	√
	Facebook	-0.0121	7.9920E-19	√
	HepPH	-0.0543	4.3067E-45	√
	CondMAT	-0.0244	2.4126E-24	√

In these tables, the first column contains the various types of FSLA under investigation. The second column lists the datasets used. The test result for each pair of FSLA types (x, y) is summarized in the third column. The column labelled as "Performance" indicates whether the FSLA of type x outperforms the FSLA of type y in terms of difference significance and mean KS distance or not with symbols "√" and "×", respectively.

The FSLA of type x statistically performs better than the FSLA of type y if the mean KS distance ($KS_x - KS_y$) is a negative value and difference significance is smaller than 0.05. As it is shown in Tables 3, 4 and 5, the $G_{2N,2}$ automaton outperforms two other automata and the Krinsky automaton performs the worst.

Table 4. The results of statistical test for different types of FSLA in terms of the KS distance for clustering coefficient distribution

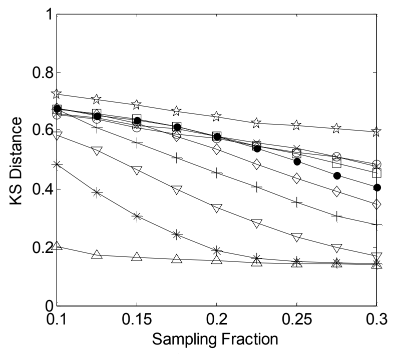
FSLA Types	Dataset	Test Result		
		Mean KS Distance ($KS_x - KS_y$)	Difference Significance	Performance
$x = L_{2N,2}$, $y = G_{2N,2}$	Flickr	0.0003	3.0914E-4	×
	Facebook	0.0013	2.9581E-7	×
	HepPH	0.0158	8.2244E-16	×
	CondMAT	0.0050	2.0310E-10	×
$x = L_{2N,2}$, $y =$ Krinsky	Flickr	-0.0056	1.5620E-8	√
	Facebook	-0.0094	4.6673E-10	√
	HepPH	-0.0113	1.0356E-11	√
	CondMAT	-0.0165	1.1411E-13	√
$x = G_{2N,2}$, $y =$ Krinsky	Flickr	-0.0073	7.1134E-10	√
	Facebook	-0.0107	4.4419E-13	√
	HepPH	-0.0272	5.6713E-26	√
	CondMAT	-0.0215	8.3377E-17	√

Table 5. The results of statistical test for different types of FSLA in terms of the KS distance for path length distribution

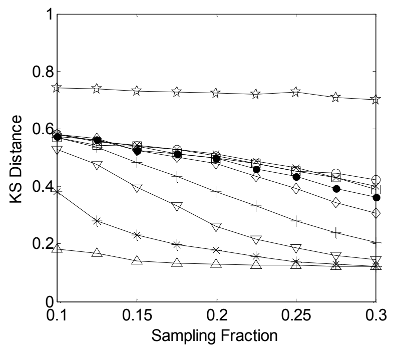
FSLA Types	Dataset	Test Result		
		Mean KS Distance ($KS_x - KS_y$)	Difference Significance	Performance
$x = L_{2N,2}$, $y = G_{2N,2}$	Flickr	0.0003	3.0914E-4	×
	Facebook	0.0013	2.9581E-7	×
	HepPH	0.0158	8.2244E-16	×
	CondMAT	0.0050	2.0310E-10	×
$x = L_{2N,2}$, $y =$ Krinsky	Flickr	-0.0056	1.5620E-8	√
	Facebook	-0.0094	4.6673E-10	√
	HepPH	-0.0113	1.0356E-11	√
	CondMAT	-0.0165	1.1411E-13	√
$x = G_{2N,2}$, $y =$ Krinsky	Flickr	-0.0073	7.1134E-10	√
	Facebook	-0.0107	4.4419E-13	√
	HepPH	-0.0272	5.6713E-26	√
	CondMAT	-0.0215	8.3377E-17	√

EXPERIMENT II

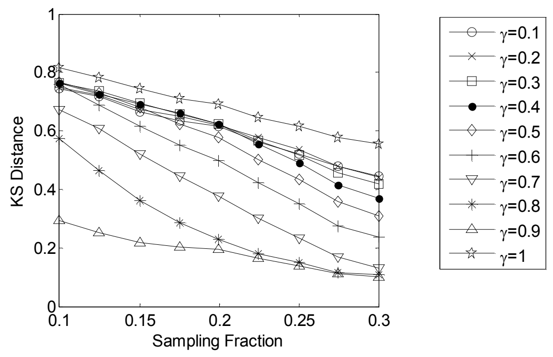
In the second experiment, we investigate the impact of the parameter γ on the performance of FLAS when it uses the $G_{2N,2}$ automaton with the parameter $N = 6$ (the best type of FSLA as reported in the previous experiment). The parameter γ varies from 0.1 to 1 with step 0.1. For each value of γ, the KS distance for different sampling fractions f is reported. Figures Fig. **11**, Fig. **12**, Fig. **13** and Fig. **14** depict the results for degree, clustering coefficient and path length distributions on Flickr, Facebook, HepPH and CondMAT datasets, respectively. As indicated, the best performance of FLAS is obtained for $\gamma = 0.9$ and the worst performance for $\gamma = 1$, on all the datasets. In our proposed algorithm, the more the value of the parameter γ, the more strictly new nodes will be added to the set V_s of sampled nodes and the more the probability that nodes with high activity are sampled (see Section 4 for more details). However, the setting $\gamma = 1$ prevents the set V_s to be updated and thus V_s of initial sample subgraph will remain unchanged while the graph is streaming (Note that in this situation, new streaming edges occurring between sampled nodes will be added to the subgraph). For this reason, FLAS performs the worst in this setting. In contrast, by considering γ equal to 0.9 the FLAS algorithm permits the possibility that new nodes can be added to the initial node set V_s, while giving the highest probability to sampling high degree nodes.



(a) Degree

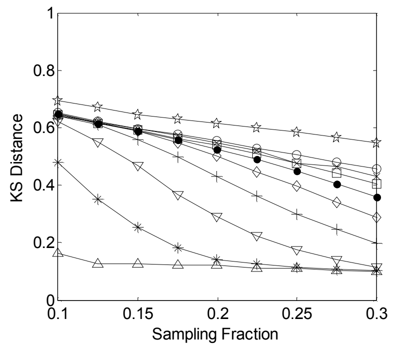


(b) Clustering Coefficient

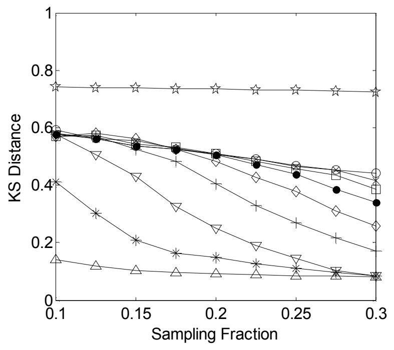


(c) Path Length

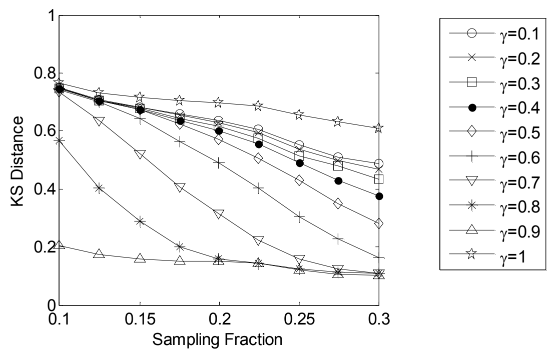
Fig. 11. KS distance for different values of *γ* on Flickr dataset.



(a) Degree

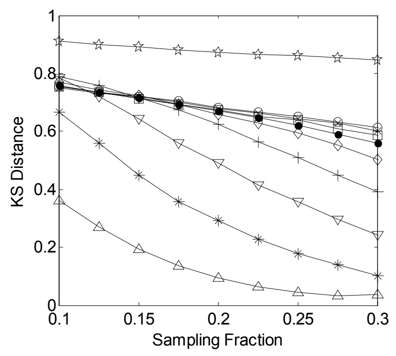


(b) Clustering Coefficient

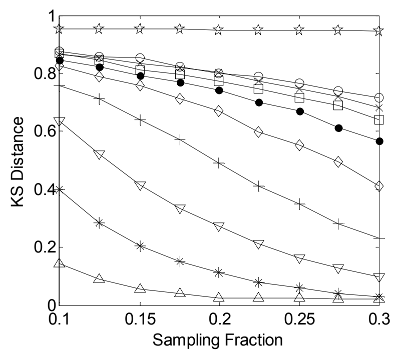


(c) Path Length

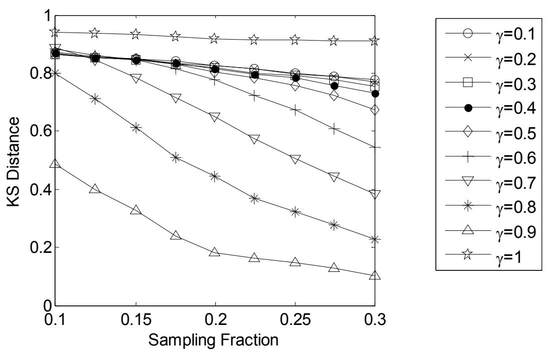
Fig. 12. KS distance for different values of *γ* on Facebook dataset.



(a) Degree

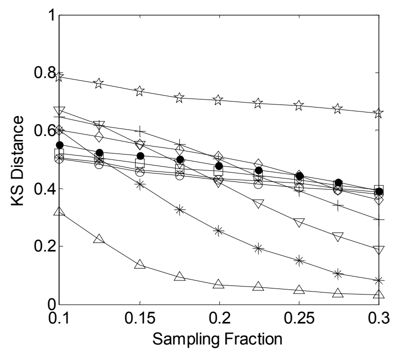


(b) Clustering Coefficient

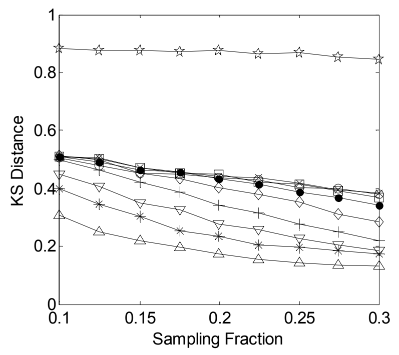


(c) Path Length

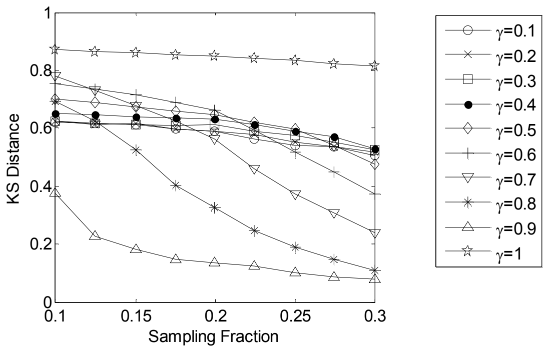
Fig. 13. KS distance for different values of γ on HepPH dataset.



(a) Degree



(b) Clustering Coefficient

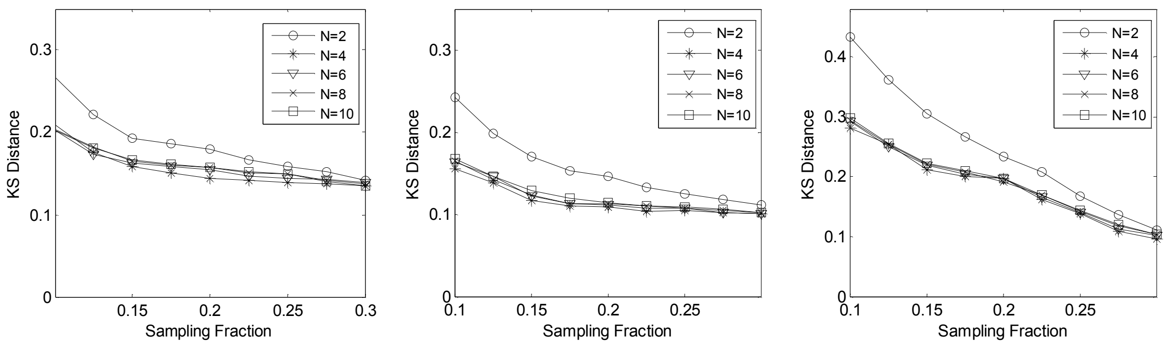


(c) Path Length

Fig. 14. KS distance for different values of γ on CondMAT dataset.

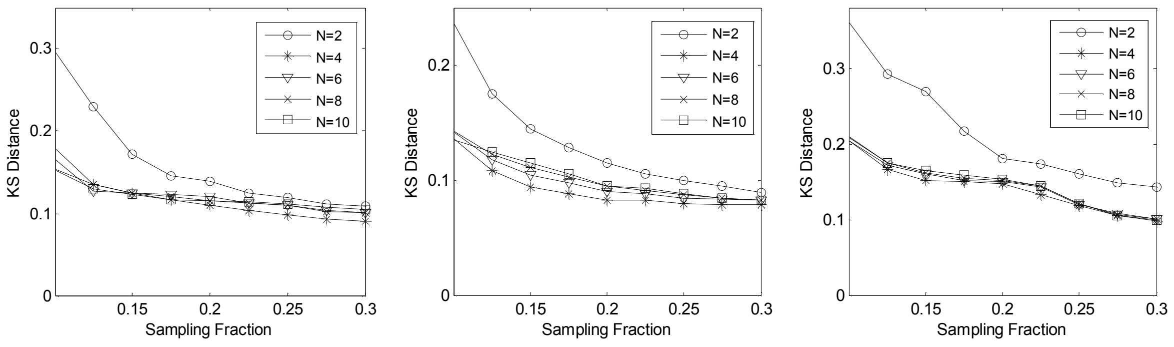
EXPERIMENT III

The last experiment of the first group of experiments aims to study the impact of the depth of memory N on the KS distance when the algorithm uses the $G_{2N,2}$ automaton. In this experiment, N varies from 2 to 10 with step 2, γ is set to 0.9 and the sampling fraction f varies from 0.1 to 0.3 with increment 0.025. Figures Fig. **15**, Fig. **16**, Fig. **17** and Fig. **18** show the KS distance for degree, clustering coefficient and path length distributions of Flickr, Facebook, HepPH and CondMAT datasets, respectively. From these figures, we can see that the best result for all datasets is obtained when $N = 4$. For this reason, for the experiments that follow we set the depth of memory N to 4.



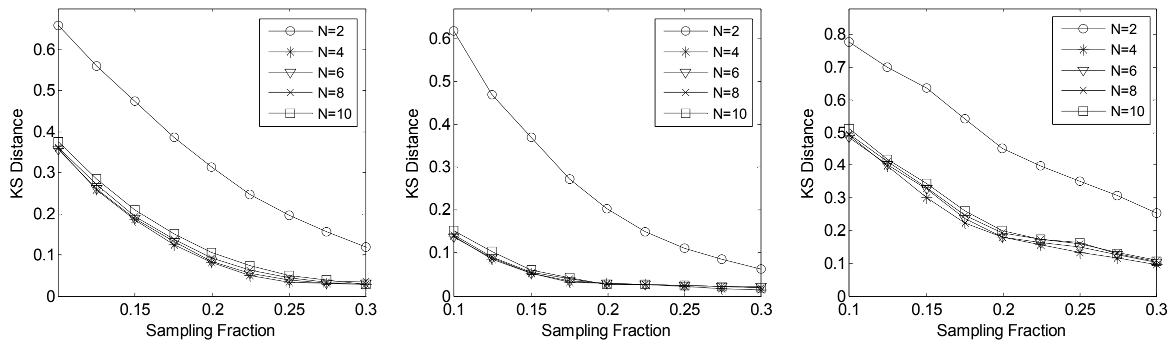
(a) Degree (b) Clustering Coefficient (c) Path Length

Fig. 15. KS distance for different values of *N* on Flickr dataset.



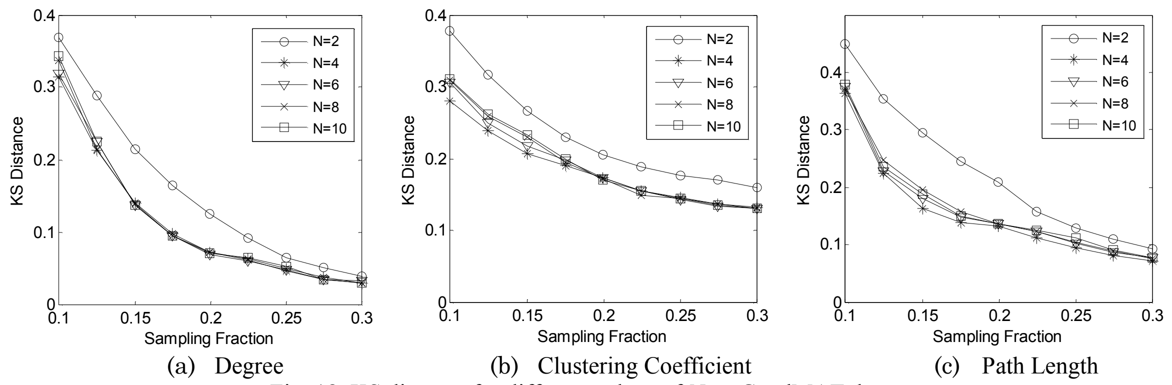
(a) Degree (b) Clustering Coefficient (c) Path Length

Fig. 16. KS distance for different values of *N* on Facebook dataset.



(a) Degree (b) Clustering Coefficient (c) Path Length

Fig. 17. KS distance for different values of *N* on HepPH dataset.



(a) Degree (b) Clustering Coefficient (c) Path Length

Fig. 18. KS distance for different values of *N* on CondMAT dataset.

EXPERIMENT IV

In the first experiment of the second group of the experiments, we compare the proposed algorithm FLAS with the PIES algorithm in terms of KS distance for degree, clustering coefficient and path length distributions. In this experiment, we use the $G_{2N,2}$ automaton with the best setting of the parameters obtained from the first group of experiments (i.e. $\gamma = 0.9$ and $N = 4$). We perform t-Test at the 95% confidence interval and report the results for the sampling fraction 0.2 in Tables 6, 7 and 8. According to the results of these tables, FLAS has significantly better performance than PIES.

Table 6. The results of statistical test for different algorithms in terms of the KS distance for degree distribution

Dataset	Test Result		
	Mean KS Distance ($KS_{FLAS} - KS_{PIES}$)	Difference Significance	Performance
Flickr	-0.0381	2.4367E-81	√
Facebook	-0.1987	8.0224E-48	√
HepPH	-0.4377	3.8966E-21	√
CondMAT	-0.3096	6.7916E-57	√

Table 7. The results of statistical test for different algorithms in terms of the KS distance for clustering coefficient distribution

Dataset	Test Result		
	Mean KS Distance ($KS_{FLAS} - KS_{PIES}$)	Difference Significance	Performance
Flickr	-0.0511	3.5431E-76	√
Facebook	-0.2054	1.2692E-46	√
HepPH	-0.4457	1.4130E-10	√
CondMAT	-0.2549	2.1055E-51	√

Table 8. The results of statistical test for different algorithms in terms of the KS distance for path length distribution

Dataset	Test Result		
	Mean KS Distance ($KS_{FLAS} - KS_{PIES}$)	Difference Significance	Performance
Flickr	-0.1066	1.2893E-68	√
Facebook	-0.1462	4.7106E-53	√
HepPH	-0.1689	2.0248E-49	√
CondMAT	-0.1744	9.3359E-41	√

EXPERIMENT V

In this experiment, we investigate the ability of the proposed algorithm FLAS in preserving the graph statistics as compared to the PIES algorithm. We consider the statistics degree, clustering coefficient, k–core and path length distributions and report the average KS distance over four datasets for different sampling fractions in Figures Fig. 19 $(a - d)$. We also plot the average L1 and L2 distances respectively for eigenvalues in Fig. Fig. 19(e) and for network values in Fig. Fig. 19(f). From these figures, we can see that the FLAS algorithm performs considerably better than the PIES algorithm for all the statistics. Tables 9 and 10 respectively show the KS distance and L1/L2 distances for each dataset at the sampling faction 0.2.

Finally, we study the ability of the FLAS algorithm in preserving the local density in sampled subgraphs comparing to the PIES algorithm. This is done by computing the maximum core numbers (the maximum value of k in the k–core distribution) for subgraphs sampled by both FLAS and PIES algorithms and comparing them with the real maximum core number for each dataset. As shown in Table 11, subgraphs sampled by FLAS have the maximum core number very closer to that of the original graphs while in PIES such phenomena does not occur.

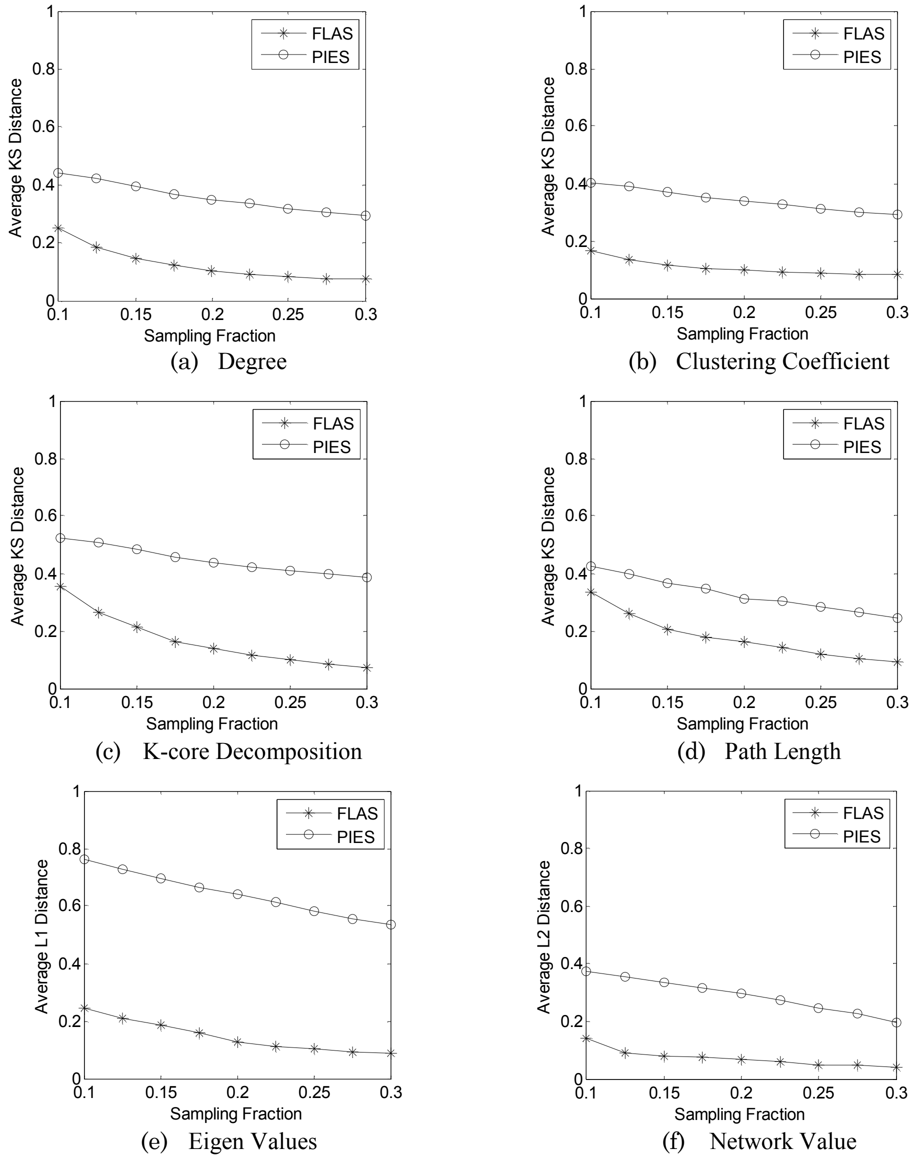


Fig. 19. (a-d) average KS distance, (e) average L_1, and (f) L_2 distance, across 4 datasets.

Table 9. KS distance for all datasets, at sampling fraction 0.2

Dataset	FLAS				PIES			
	Deg	Clust	k Core	Path	Deg	Clust	k Core	Path
Flickr	0.1445	0.1087	0.1721	0.1963	0.1826	0.1598	0.2145	0.3029
Facebook	0.1099	0.0829	0.1226	0.1471	0.3086	0.2883	0.3846	0.2933
HepPH	0.0813	0.0288	0.1697	0.1786	0.5190	0.4745	0.6641	0.3475
CondMAT	0.0720	0.1726	0.0858	0.1323	0.3816	0.4275	0.4829	0.3067
Avg. for all datasets	**0.1019**	**0.0983**	**0.1376**	**0.1636**	0.3479	0.3375	0.4365	0.3126

Table 10. L_1/L_2 distance for all datasets, at sampling fraction 0.2

Dataset	FLAS		PIES	
	Eigen Val	Net Val	Eigen Val	Net Val
Flickr	0.0071	0.0013	0.5496	0.1923
Facebook	0.1266	0.0759	0.6285	0.3435
HepPH	0.2147	0.1203	0.7151	0.3383
CondMAT	0.1605	0.0768	0.6743	0.3172
Avg. for all datasets	**0.1272**	**0.0686**	0.6419	0.2978

Table 11. The maximum core number for sampling fraction 0.2 versus its real value for each dataset

Dataset	Real Max Core no.	FLAS	PIES
Flickr	406	**221**	162
Facebook	16	**12**	5
HepPH	30	**20**	5
CondMAT	25	**16**	6

EXPERIMENT VI

This experiment plots the distributions of six graph statistics for each of the four datasets when sampling fraction f is set to 0.2. For degree and k–core distributions, CCDF (complementary cumulative distribution function) is plotted and for clustering coefficient and path length distributions, we plot CDF. As shown in Figures Fig. **20**, Fig. **21**, Fig. **22** and Fig. **23**, FLAS preserves the degree distribution more accurately than PIES. The PIES algorithm underestimates the degree distribution for all datasets, except for Flickr. PIES also underestimates the clustering coefficient distribution for Flickr, CondMAT and HepPH. The FLAS algorithm captures the clustering coefficient statistic for HepPH, underestimates it for CondMAT, and overestimates it for Flickr and Facebook.

For the k–core distribution, FLAS generally provides better results compared to the PIES algorithm. While FLAS preserves the k–core distribution in almost all the datasets, PIES overestimates the core structures in Flickr and underestimates them in the other datasets. FLAS captures the path length distribution for all the tested graphs. However, PIES underestimates this statistic for Flickr and overestimates it for *CondMAT* and *HepPH.* For Facebook graph, PIES performs almost as good as FLAS. FLAS accurately estimates the eigenvalues and the network values of Flicker graph and for the other graphs, it outperforms the PIES algorithm for these two statistics.

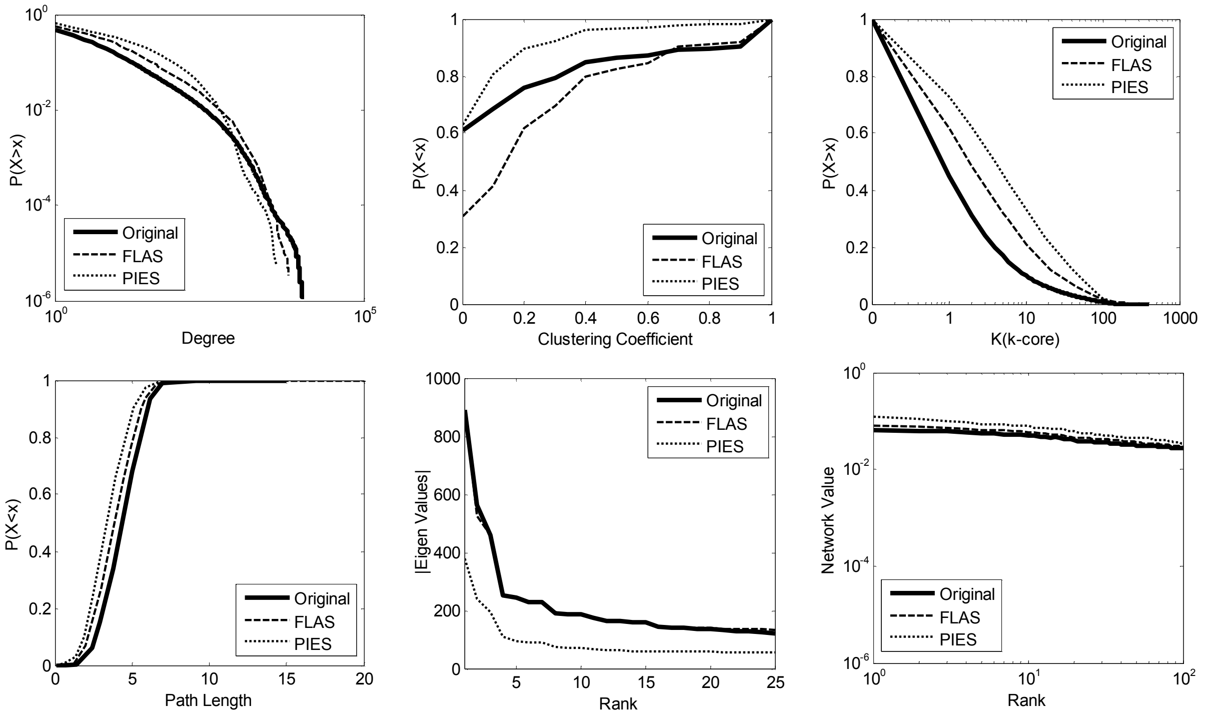


Fig. 20. Comparing sampling algorithms on Flickr dataset for different distribution statistics, at sampling fraction 0.2.

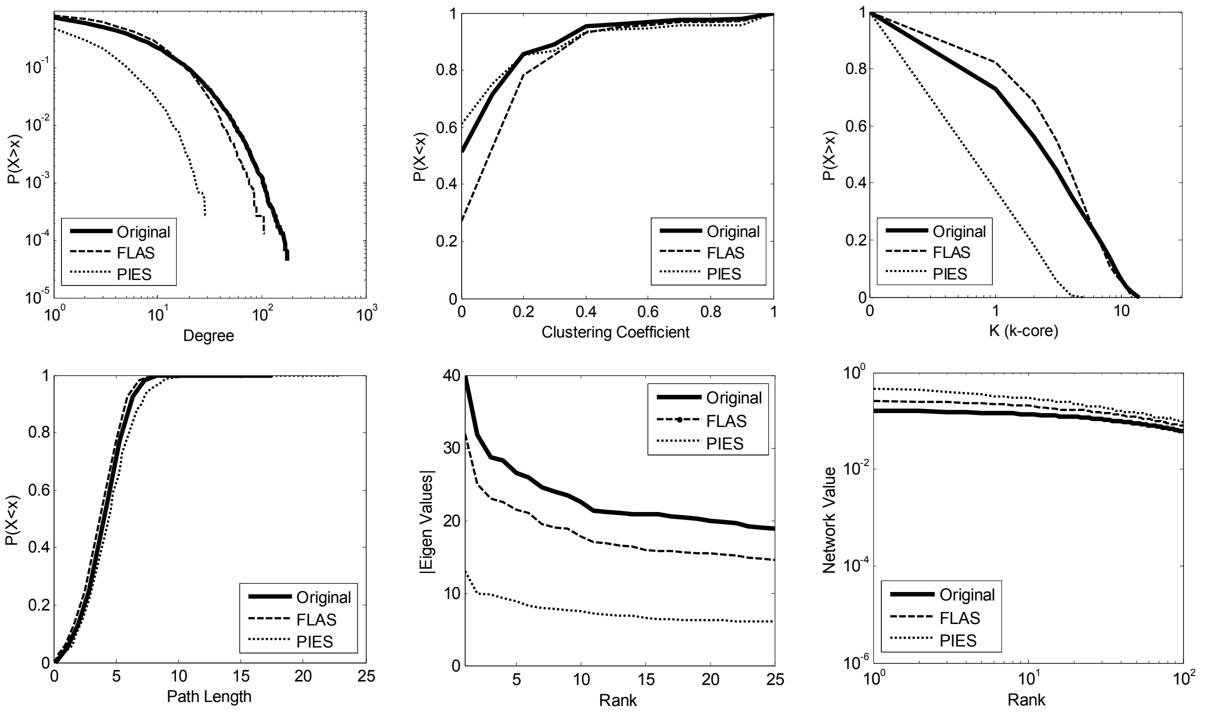


Fig. 21. Comparing sampling algorithms on Facebook dataset for different distribution statistics, at sampling fraction 0.2.

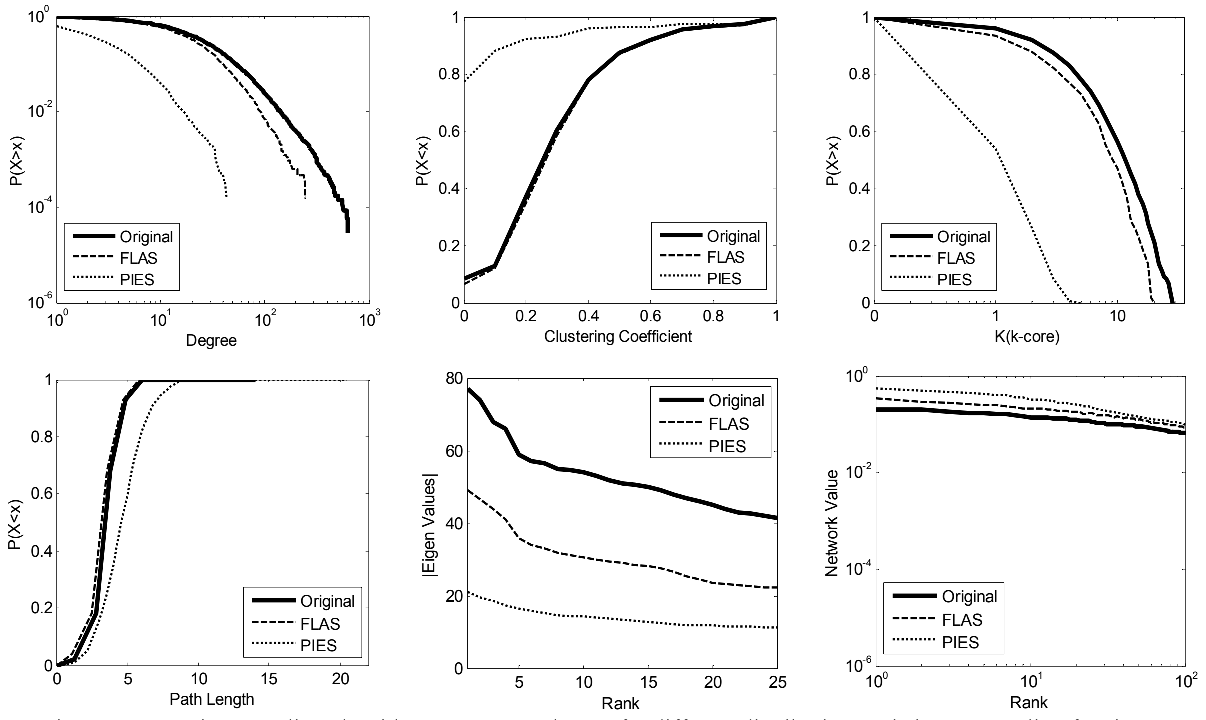


Fig. 22. Comparing sampling algorithms on HepPH dataset for different distribution statistics, at sampling fraction 0.2.

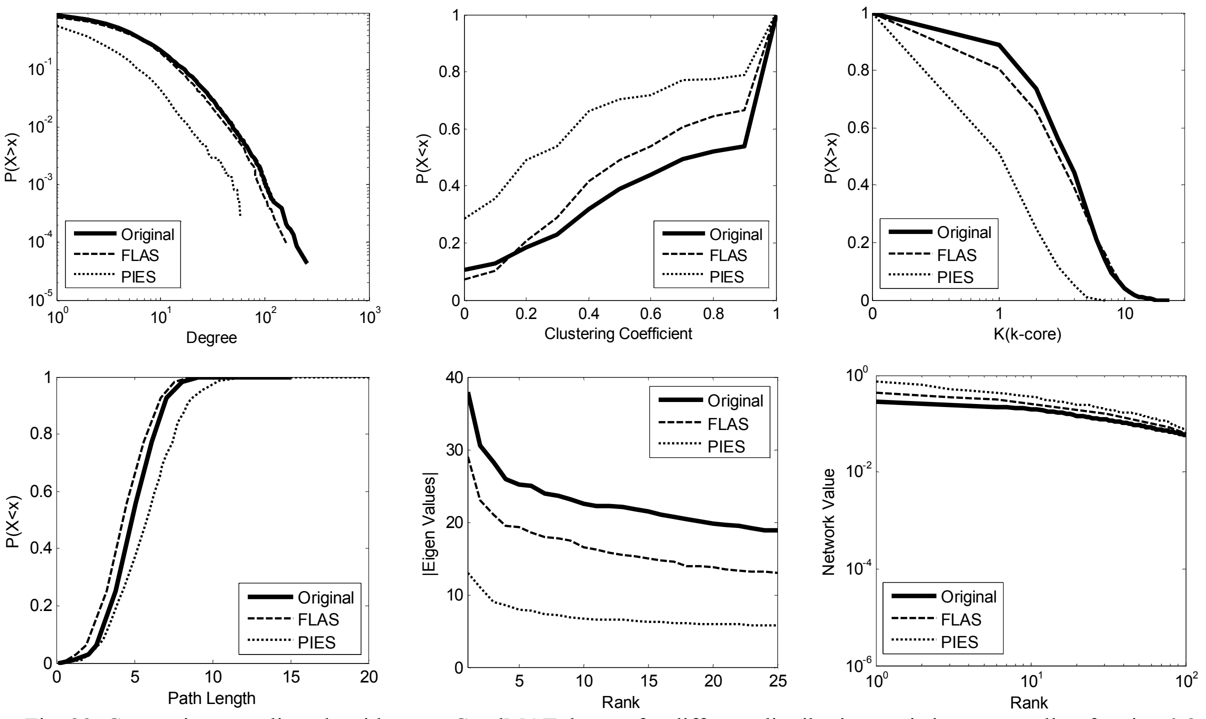


Fig. 23. Comparing sampling algorithms on CondMAT dataset for different distribution statistics, at sampling fraction 0.2.

EXPERIMENT VII

In this experiment, we compare the number of isolated nodes, connected components, and the 100 highest degree nodes (top 100 hubs) in subgraphs sampled by the proposed algorithm FLAS and PIES. We consider the sampling fraction of 0.2 and report the probability of isolated nodes, as well as the proportion of the top 100 hubs accumulated by sample subgraphs for the both algorithms in Table 12. We also compare the number of connected components in subgraphs sampled by both FLAS and PIES at the sampling fraction 0.2 with the real number of them for each dataset in Table 13. From these tables, we can see that, as discussed in subsection 4.1, subgraphs sampled by PIES have the higher probability of isolated nodes and include the less proportion of the highest degree nodes as comparing to FLAS. In FLAS, the probability of sampling an edge depends on previous streaming edges and the nodes with high activity have a higher probability of inclusion in sampled subgraphs. As a result, FLAS produces more connected subgraphs than PIES. As reported in Table 13, subgraphs sampled by FLAS have the less number of connected components than those sampled by PIES. Clearly, the less the number of disjoint connected components, the more the connectivity is in graph.

Table 12. The probability of isolated nodes and the proportion of hubs, at sampling fraction 0.2

Dataset	Isolated Node		Hub (Top 100)	
	FLAS	PIES	FLAS	PIES
Flickr	**0.1301**	0.1623	**0.59**	0.20
Facebook	**0.1257**	0.1605	**0.70**	0.21
HepPH	**0.0424**	0.0970	**0.72**	0.68
CondMAT	**0.1074**	0.1514	**0.82**	0.55

Table 13. The number of connected components for sampling fraction 0.2 versus its real value for each dataset

Dataset	Real Connected Components	FLAS	PIES
Flickr	1	**919**	21668
Facebook	842	**108**	1096
HepPH	61	**27**	316
CondMAT	567	**127**	340

EXPERIMENT VIII

This experiment aims to study the performance of the proposed sampling algorithm FLAS over time while the original graph is streaming as compared to PIES. For this purpose, at first an initial subgraph is sampled from the original graph. Then, it is investigated that how the quality of the sample subgraph changes after reviewing different proportions of the remaining edges of the stream. We consider the sampling fraction 0.2 for the initial subgraph. The reviewing proportion varies from 0 to 30% and for each proportion, we report the average KS distance over four datasets for the statistics degree, clustering coefficient and path length distributions in Figures Fig. 24$(a - c)$. In these figures, the proportion of 0 refers to the initial subgraph when no remaining edge of the stream has been reviewed.

As one can see, increasing the proportion of reviewed edges results in the decrease of average KS distance for both FLAS and PIES. However, this decrease is faster for FLAS such that for a same proportion, FLAS always provides much better results than PIES. In this experiment, we also investigate the performance of FLAS when a stop condition is considered for it, where updating the sample continues until all the learning automata corresponding to the nodes in V_s are in their most internal state. In Table 14, we report the average proportion of edges that are reviewed and the average KS distance that is obtained for three graph statistics by FLAS with and without using the stop condition. Based on the results of this table,

Table 14. Average reviewing proportion and KS distance across all datasets for FLAS, at sampling fraction 0.2FLAS meets the stop condition on all datasets after reviewing on average 69% of the remaining edges of the stream and obtains the results close to when there is no stop condition considered in FLAS (i.e. all the streaming edges are reviewed). Therefore, one can use the sampling algorithm FLAS with the stop condition and reduce the computational cost of the algorithm, while obtaining subgraphs of comparable quality to those produced by FLAS without the stop condition.

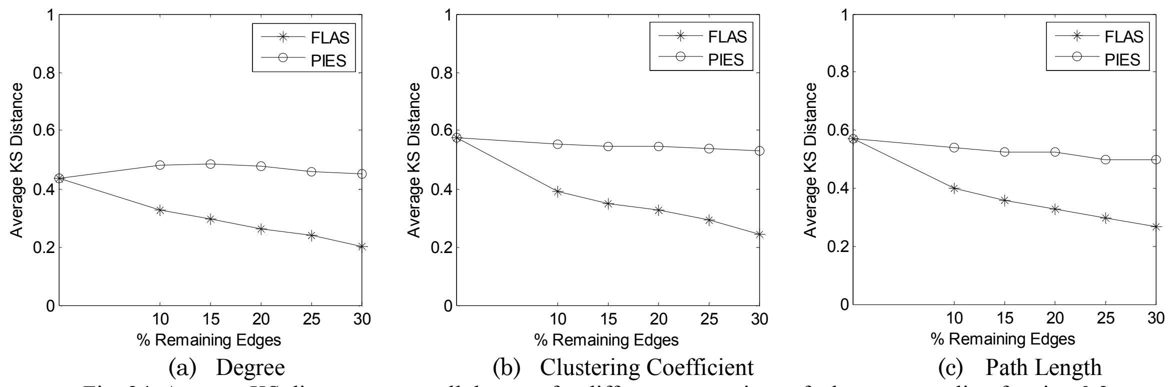


Fig. 24. Average KS distance across all datasets for different proportions of edges, at sampling fraction 0.2.

Table 14. Average reviewing proportion and KS distance across all datasets for FLAS, at sampling fraction 0.2

FLAS	% Remaining Edges	Deg	Clust	Path
Without stop condition	100	0.1019	0.0983	0.1636
With stop condition	69	0.1323	0.1302	0.1796

EXPERIMENT IX

In this experiment, we validate the efficiency of using the high degree bias for offsetting the downward bias in the degree distribution of subgraphs sampled from scale-free networks. We generate four synthetic networks that we refer to them as: WS_Net$_1$, WS_Net$_2$, BA_Net$_1$ and BA_Net$_2$, where two first networks are small-world networks built according to the Watts-Strogatz model [52], and two latter networks are scale-free networks based on the Barabasi-Albert model [45]. Table 15 provides some information about these synthetic networks. For the sampling fraction of 0.2, we plot the degree distribution for each of the synthetic networks as given in Figure Fig. **25**. From this figure, we observe that, as mentioned in subsection 4.1, using the high degree bias help the both algorithms to preserve more accurately the degree distribution of the scale-free networks. However, this technique is not appropriate for the small-world networks, since the both FLAS and PIES fail to capture the degree distribution of these networks. It is worthy to note that FLAS always performs better than PIES, especially on the scale-free networks.

Table 15. Characteristics and parameters of synthetic networks

Graph	Nodes	Edges	Density	Parameters
WS-Net$_1$	30000	90000	2E-4	k=6, P=0.2
WS-Net$_2$	20000	100000	5E-4	k=10, P=0.2
BA-Net$_1$	30000	149987	3.3E-4	m0=10, m=5
BA-Net$_2$	20000	99985	4.9E-4	m0=6, m=5

6. Discussion And Conclusion

This paper addressed the problem of sampling from activity networks in which the stream of edges continuously evolves over time. These networks are highly dynamic and include a massive volume of edges. Most previous work on sampling from networks either has assumed the network graph is static and fully accessible at any step, or despite considering the stream evolution has not addressed the problem of sampling a representative subgraph from the original graph. To the best of our knowledge, the algorithm PIES is the best streaming sampling method reported so far which considers the dynamicity and hug size of streaming networks and produces subgraphs preserving the underlying properties of these networks. Despite its performance, PIES suffers two drawbacks: independent sampling of edges and the random replacement of sampled nodes. As a result of these drawbacks, this method cannot preserve the original connectivity and performs well only for sparse, less clustered graphs.

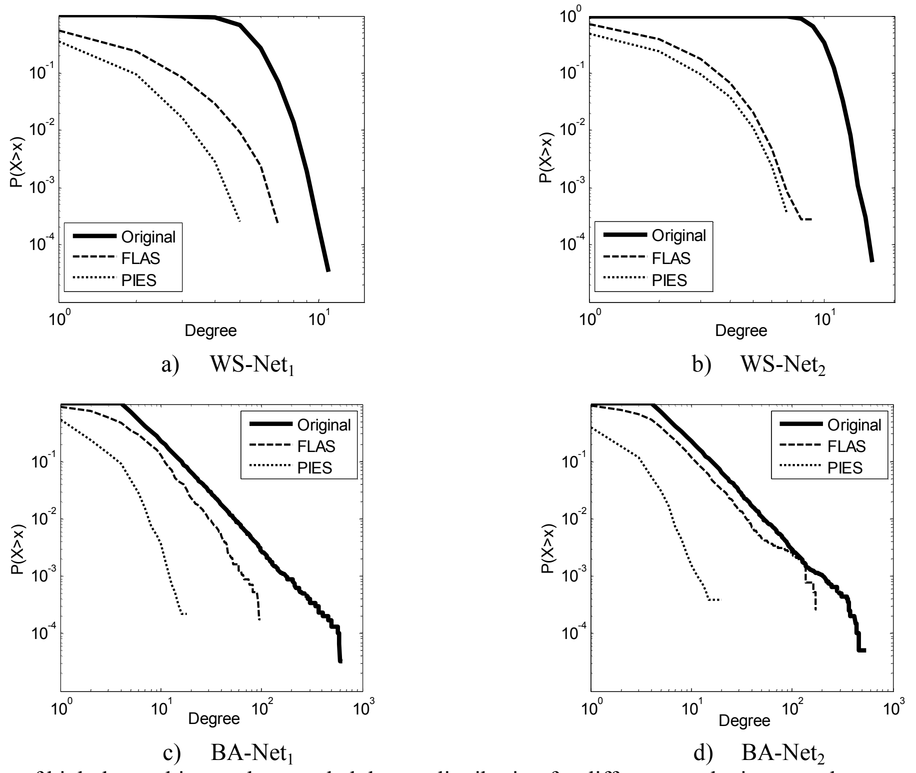


Fig. 25. The impact of high degree bias on the sampled degree distribution for different synthetic networks, at sampling fraction 0.2.

In this paper, we proposed a new streaming sampling algorithm based on fixed structure learning automata which overcomes PIES's drawbacks, while maintaining the advantages of this method. The proposed algorithm satisfies the both goals of being implemented in a streaming fashion and producing representative samples. FLAS adds an edge to the sample and chooses sampled nodes for the replacement on the bases of the decisions made by the learning automata assigned to the nodes of the original graph. In this way, the probability of sampling an edge depends on previous observed edges and the sampled nodes with low activity have a higher probability to be replaced.
In order to evaluate the performance of our proposed algorithm, we conducted two groups of extensive experiments on real-world activity network datasets and synthetic networks. The experiments were done in terms of Kolmogorov-Smirnov (KS) test for degree, clustering coefficient, k–core and path length distributions and also in terms of normalized L_1 and L_2 distances respectively for eigenvalues and network values. The first group of experiments was dedicated to finding effective settings for the parameters of the proposed algorithm FLAS. From these experiments, we observed that using the $G_{2N,2}$ automaton with the memory depth of 4 provides the best results in comparison with other settings. We also found that by considering the probability γ equal to 0.9 the algorithm FLAS preserves more accurately three distributions of degree, clustering coefficient and path length for all test datasets. The reason is that by this setting the FLAS is more probable to sample high degree nodes and thus produce subgraphs with more connectivity.
In the second group of experiments, we compared the performance of our algorithm FLAS with the algorithm PIES. The empirical results indicated that the proposed algorithm significantly outperforms PIES in terms of the ability to produce representative samples. In particular, based on the results of experiments IV, V and VI, subgraphs sampled by FLAS provide the less KS distance and capture better the original distributions of six graph statistics, namely degree, clustering coefficient, k–core, path length, eigenvalues, and network values. The reason is that FLAS produces more connected subgraphs which preserve more accurately the topological properties of original graphs [1]. In contrary, because of its two drawbacks, the

algorithm PIES has the higher probability of isolated nodes, and includes the less proportion of the highest degree nodes and the more number of disjoint connected components in its sampled subgraphs as comparing to FLAS (for more details, see experiment VII). We also investigated that how the quality of subgraph sampled by FLAS changes over time while the original graph is streaming and when this sample updating process can be stopped. We found that the performance of FLAS is improved much faster than that of PIES along time while reviewing new edges. We also found that FLAS can stop the updating process after reviewing 69% of streaming edges and obtain subgraphs of acceptable quality. Finally, we validated the ability of high degree sampling for offsetting the downward bias in the degree distribution of subgraphs sampled from scale-free networks [4, 11].

Summing up, using fixed structure learning automata helps our proposed algorithm FLAS to overcome the drawbacks of the algorithm PIES. As a result, FLAS not only provides good results on sparse, less clustered graphs, but also can produce high quality subgraphs for dense and high clustered graphs.

ACKNOWLEDGEMENT

The authors acknowledge the use of high performance computers provided by High Performance Computing Research Center (HPCRC) at Amirkabir University of Technology, in the completion of this work.

REFERENCES

1. Leskovec J, Faloutsos C (2006) Sampling from large graphs. In: Proc. 12th ACM SIGKDD Int. Conf. Knowl. Discov. data Min. - KDD '06. ACM Press, New York, New York, USA, p 631
2. Ebbes P, Huang Z, Rangaswamy A (2012) Subgraph Sampling Methods for Social Networks: The Good, the Bad, and the Ugly. SSRN Electron J. doi: 10.2139/ssrn.1580074
3. Lee SH, Kim P-J, Jeong H (2006) Statistical properties of sampled networks. Phys Rev E 73:16102. doi: 10.1103/PhysRevE.73.016102
4. Yoon S, Lee S, Yook S-H, Kim Y (2007) Statistical properties of sampled networks by random walks. Phys Rev E 75:46114. doi: 10.1103/PhysRevE.75.046114
5. Ghavipour M, Meybodi MR (2017) Irregular cellular learning automata-based algorithm for sampling social networks. Eng Appl Artif Intell 59:244–259.
6. Krishnamurthy V, Faloutsos M, Chrobak M, et al (2007) Sampling large Internet topologies for simulation purposes. Comput Networks 51:4284–4302. doi: 10.1016/j.comnet.2007.06.004
7. Hübler C, Kriegel H-P, Borgwardt K, Ghahramani Z (2008) Metropolis Algorithms for Representative Subgraph Sampling. In: 2008 Eighth IEEE Int. Conf. Data Min. IEEE, pp 283–292
8. Kurant M, Markopoulou A, Thiran P (2011) Towards Unbiased BFS Sampling. IEEE J Sel Areas Commun 29:1799–1809. doi: 10.1109/JSAC.2011.111005
9. Rezvanian A, Meybodi MR (2015) Sampling social networks using shortest paths. Phys A Stat Mech its Appl 424:254–268. doi: 10.1016/j.physa.2015.01.030
10. Rezvanian A, Meybodi MR (2015) A new learning automata-based sampling algorithm for social networks. Int J Commun Syst n/a-n/a. doi: 10.1002/dac.3091
11. Ahmed NK, Neville J, Kompella R (2014) Network Sampling: From Static to Streaming Graphs. ACM Trans Knowl Discov Data 8:7. doi: 10.1145/2601438
12. Bar-Yossef Z, Kumar R, Sivakumar D (2002) Reductions in streaming algorithms, with an application to counting triangles in graphs. In: Proc. Thirteen. Annu. ACM-SIAM Symp. Discret. algorithms. Society for Industrial and Applied Mathematics, San Francisco, California, pp 623–632
13. Aggarwal CC (2006) On biased reservoir sampling in the presence of stream evolution. Proc 32nd Int Conf Very large data bases 607–618.
14. Sarma A Das, Gollapudi S, Panigrahy R (2011) Estimating PageRank on graph streams. J ACM 58:1–19. doi: 10.1145/1970392.1970397
15. Buriol LS, Frahling G, Leonardi S, et al (2006) Counting triangles in data streams. In: Proc. twenty-fifth ACM SIGMOD-SIGACT-SIGART Symp. Princ. database Syst. ACM, pp 253–262
16. Aggarwal CC, Li Y, Yu PS, Jin R (2010) On dense pattern mining in graph streams. Proc VLDB Endow 3:975–984.
17. Aggarwal CC, Zhao Y, Yu PS (2010) On clustering graph streams. In: Proc. 2010 SIAM Int. Conf. Data Min. SIAM, pp 478–489
18. Chen L, Wang C (2010) Continuous subgraph pattern search over certain and uncertain graph streams. IEEE Trans Knowl Data Eng 22:1093–1109.

19. Cormode G, Muthukrishnan S (2005) Space efficient mining of multigraph streams. In: Proc. twenty-fourth ACM SIGMOD-SIGACT-SIGART Symp. Princ. database Syst. - Pod. ’05. ACM Press, New York, New York, USA, p 271
20. Ahmed NK, Berchmans F, Neville J, Kompella R (2010) Time-based sampling of social network activity graphs. In: Proc. Eighth Work. Min. Learn. with Graphs - MLG ’10. ACM Press, New York, New York, USA, pp 1–9
21. Aggarwal CC, Zhao Y, Philip SY (2011) Outlier detection in graph streams. In: Data Eng. (ICDE), 2011 IEEE 27th Int. Conf. IEEE, pp 399–409
22. Jin EM, Girvan M, Newman MEJ (2001) Structure of growing social networks. Phys Rev E 64:46132.
23. Tang L, Liu H (2010) Community detection and mining in social media. Synth Lect Data Min Knowl Discov 2:1–137.
24. Leskovec J, Kleinberg J, Faloutsos C (2005) Graphs over time. In: Proceeding Elev. ACM SIGKDD Int. Conf. Knowl. Discov. data Min. - KDD ’05. ACM Press, New York, New York, USA, p 177
25. Kumar R, Novak J, Tomkins A (2010) Structure and evolution of online social networks. In: Link Min. Model. algorithms, Appl. Springer, pp 337–357
26. Stumpf MP, Wiuf C, May RM (2005) Subnets of scale-free networks are not scale-free: sampling properties of networks. In: Proc. Natl. Acad. Sci. U. S. A. National Acad Sciences, pp 4221--4224
27. Ahn Y-Y, Han S, Kwak H, et al (2007) Analysis of topological characteristics of huge online social networking services. In: Proc. 16th Int. Conf. World Wide Web. ACM, pp 835–844
28. Mislove A, Marcon M, Gummadi KP, et al (2007) Measurement and analysis of online social networks. In: Proc. 7th ACM SIGCOMM Conf. Internet Meas. ACM, pp 29–42
29. Wilson C, Boe B, Sala A, et al (2009) User interactions in social networks and their implications. In: Proc. 4th ACM Eur. Conf. Comput. Syst. Acm, pp 205–218
30. Goodman LA (1961) Snowball Sampling. Ann Math Stat 32:148–170.
31. Gjoka M, Kurant M, Butts CT, Markopoulou A (2010) Walking in Facebook: A case study of unbiased sampling of OSNs. In: Infocom, 2010 Proc. IEEE. IEEE, pp 1–9
32. Ye S, Lang J, Wu F (2010) Crawling online social graphs. In: Web Conf. (APWEB), 2010 12th Int. Asia-Pacific. IEEE, pp 236–242
33. Lu J, Li D (2012) Sampling online social networks by random walk. In: Proc. First ACM Int. Work. Hot Top. Interdiscip. Soc. Networks Res. - HotSocial ’12. ACM Press, New York, New York, USA, pp 33–40
34. Kurant M, Gjoka M, Butts CT, Markopoulou A (2011) Walking on a graph with a magnifying glass. In: Proc. ACM SIGMETRICS Jt. Int. Conf. Meas. Model. Comput. Syst. - SIGMETRICS ’11. ACM Press, New York, New York, USA, p 281
35. Rasti AH, Torkjazi M, Rejaie R, et al (2009) Respondent-driven sampling for characterizing unstructured overlays. In: INFOCOM 2009, IEEE. IEEE, pp 2701–2705
36. Lee C-H, Xu X, Eun DY, et al (2012) Beyond random walk and metropolis-hastings samplers. In: Proc. 12th ACM SIGMETRICS/PERFORMANCE Jt. Int. Conf. Meas. Model. Comput. Syst. - SIGMETRICS ’12. ACM Press, New York, New York, USA, p 319
37. Stutzbach D, Rejaie R, Duffield N, et al (2009) On unbiased sampling for unstructured peer-to-peer networks. IEEE/ACM Trans Netw 17:377–390.
38. Ribeiro B, Towsley D (2010) Estimating and sampling graphs with multidimensional random walks. In: Proc. 10th ACM SIGCOMM Conf. Internet Meas. ACM, pp 390–403
39. Avrachenkov K, Ribeiro B, Towsley D (2010) Improving random walk estimation accuracy with uniform restarts. In: Int. Work. Algorithms Model. Web-Graph. Springer, pp 98–109
40. Thathachar MAL, Sastry PS (2011) Networks of learning automata: Techniques for online stochastic optimization. Springer Science & Business Media
41. Narendra KS, Thathachar MAL (2012) Learning automata: an introduction. doi: 10.1109/TSMCB.2002.1049606
42. Ghavipour M, Meybodi MR (2016) An adaptive fuzzy recommender system based on learning automata. Electron Commer Res Appl 20:105–115.
43. Mirsaleh MR, Meybodi MR (2016) A new memetic algorithm based on cellular learning automata for solving the vertex coloring problem. Memetic Comput 8:2112–222. doi: 10.1007/s12293-016-0183-4
44. Tsetlin M (1961) On behaviour of finite automata in random medium. Avtom I Telemekhanika 22:1345--1354.
45. Barabási A-L, Albert R (1999) Emergence of scaling in random networks. Science (80-) 286:509–512.
46. Albert R, Jeong H, Barabási A-L (2000) Error and attack tolerance of complex networks. Nature 406:378–382.
47. Bayer R, McCreight E (2002) Organization and Maintenance of Large Ordered Indexes. In: Softw. Pioneers. Springer Berlin Heidelberg, Berlin, Heidelberg, pp 245–262
48. Gleich DF (2012) Graph of Flickr Photo-Sharing Social Network Crawled in May 2006. DOI 104231/D39P2W550. doi: 10.4231/D39P2W550

49. Viswanath B, Mislove A, Cha M, Gummadi KP (2009) On the evolution of user interaction in Facebook. In: Proc. 2nd ACM Work. Online Soc. networks - WOSN ’09. ACM Press, New York, New York, USA, p 37
50. Leskovec J, Krevl A (2014) SNAP Datasets:Stanford Large Network Dataset Collection.
51. Goldstein ML, Morris SA, Yen GG (2004) Problems with fitting to the power-law distribution. Eur Phys J B 41:255–258. doi: 10.1140/epjb/e2004-00316-5
52. Watts DJ, Strogatz SH (1998) Collective dynamics of “small-world”networks. Nature 393:440–442.