Space-Efficient Sampling from Social Activity Streams

Nesreen K. Ahmed, Jennifer Neville, and Ramana Kompella
Computer Science Department, Purdue University
{nkahmed,neville,kompella}@cs.purdue.edu

ABSTRACT

In order to efficiently study the characteristics of network domains and support development of network systems (*e.g.* algorithms, protocols that operate on networks), it is often necessary to *sample* a representative subgraph from a large complex network. Although recent subgraph sampling methods have been shown to work well, they focus on sampling from memory-resident graphs and assume that the sampling algorithm can access the entire graph in order to decide which nodes/edges to select. Many large-scale network datasets, however, are too large and/or dynamic to be processed using main memory (e.g., email, tweets, wall posts). In this work, we formulate the problem of sampling from large graph streams. We propose a streaming graph sampling algorithm that dynamically maintains a representative sample in a reservoir based setting. We evaluate the efficacy of our proposed methods empirically using several real-world data sets. Across all datasets, we found that our method produce samples that preserve better the original graph distributions.

1. INTRODUCTION

Many real-world complex systems can be represented as graphs and networks—from information networks, to communication networks, to biological networks. Naturally, there has been a lot of interest in studying characteristics of these networks, modeling their structure, as well as developing algorithms and systems that operate on the networks. While the recent surge in activity in online social networks (*e.g.*, Facebook, Twitter) has prompted a similar need for characterization and modeling efforts, it is often much harder than in traditional networks due to their size. Specifically, these networks tend to be too large to efficiently acquire, store and/or analyze (*e.g.*, one billion chat messages per day in Facebook [35]). It is therefore often necessary to *sample* smaller subgraphs from the larger network structure, that can then be used to investigate the characteristics and properties of the larger network. It can also be used to drive realistic simulations and experimentation before deploying new protocols and systems in the field—for example, new Internet protocols, social/viral marketing schemes, and/or fraud detection algorithms.

In this work, we consider the following graph sampling problem: Assume an input graph $G = (V, E)$ of size $N = |V|$, from which a sampling algorithm selects a subgraph $G_s = (V_s, E_s)$ with a subset of the nodes ($V_s \subset V$) and/or edges ($E_s \subset E$), such that $|V_s| = \phi N$. We refer to ϕ as the sampling fraction. The goal is to sample a *representative* subgraph G_s that matches many of the properties of G, so that G_s can be used to *simultaneously* preserve several characteristics of the network structure in the original graph G (*e.g.*, degree, path length, clustering). Specifically, we aim to select a G_s that minimizes the distributional distance over several graph measures (*e.g.*, degree distribution) simultaneously. Let $f(.)$ be a property of a graph, then the goal is to select a sample that minimizes the distance between the property in G and the property in G_s: $dist[f(G), f(G_S)]$. In this work, we consider degree, hop plot, and clustering distributions for $f(.)$ and evaluate using two distributional distance metrics—Kolmogorov-Smirnov distance and skew divergence [23].

While many graph sampling methods have been proposed before (*e.g.*, [17, 26]), they typically require access to the whole graph in its entirety at any step, in order to decide which nodes/edges to select. While the graph data can be stored on disks, processing full large graphs is usually done using physical memory (RAM) which is a limited/expensive resource. Therefore, the ideal approach to process the graph data is to use a *streaming model*, where the graph data is presented as a stream of edges, and any computation on the stream relies on using a small amount of memory, and in a single pass. Many large-scale network datasets readily admit such a streaming model. For example, online social network applications (*e.g.* Facebook, Twitter) have data that consist of micro-communications among users (*e.g.* wall posts, tweets, emails); any activity between two users can result in an edge getting added to the activity graph.

In this work, we consider the problem of sampling from such large social activity streams. We refer to social activity streams as graph streams since social activities can be represented as a graph. Specifically, *our goal* is to devise a streaming algorithm for sampling subgraphs from large graph streams, that can decide whether to include an edge in the sampled graph, as the edge is streamed in.

While there is a great deal of research on data streams and data stream management, to our knowledge, our work is the first work to focus on streaming algorithms for sampling subgraphs from large graph streams in a single pass. Satisfying the dual objective of finding a sampling algorithm that can sample representative subgraphs, while being amenable to a streaming implementation is quite challenging. Most existing sampling algorithms fail to process graph streams (*i.e.* requiring multiple passes over the edges). As an example, breadth-first search needs to access the full neighborhood of a node to perform one step of its process.

BigMine '12, August 12, 2012 Beijing, China

In this paper, we propose a novel sampling algorithm that is amenable to streaming implementations. Specifically, we propose *partially-induced edge sampling (PIES)* that randomly samples edges, induces the sampled nodes, and maintains a dynamic/changing sample in a reservoir-based setting using a single pass over the edges.

Our proposed approach is simple, efficient, and can be used to sample large graphs that are too large to fit in memory. Moreover, it can also be used to graphs that readily admit the streaming model (*e.g.* email logs, tweets between users in Twitter).

We evaluate PIES over a number of real world (*e.g.*, Facebook, Twitter, HepPH, Flickr) datasets collected by other researchers ([1, 39]), and an email network constructed from two weeks of Purdue University email traffic. We compare PIES to existing/proposed baseline stream sampling techniques such as edge sampling, node sampling and a simple breadth-first search (BFS) based algorithm.

Across all datasets, we observed that PIES produces samples that better match the distributions of degree, path length and clustering compared to other existing algorithms.

The rest of the paper is organized as follows. We first present a background and related work on sampling methods in section 2 . Next, we outline our proposed sampling algorithms with streaming implementations in section 3. Finally, we compare PIES with other baseline graphs sampling algorithms in section 4.

2. GRAPH SAMPLING ALGORITHMS

In this section, we discuss standard graph sampling algorithms in literature, which can be broadly classified as node-based, edge-based, and topology-based methods. Most graph sampling algorithms have two basic components: (1) node selection, and (2) subgraph formation. The node selection step identifies a sample set of nodes (V_s), while the subgraph formation step selects the set of edges (E_s) to be included in the sampled subgraph. We distinguish between two different approaches to subgraph formation—total and partial graph induction—which differ by whether *all* or *some* of the edges incident on the sampled nodes are included in the sampled graph. The resulting sampled graphs are referred to as the *induced subgraph* and *partially induced subgraph* respectively.

Node sampling (NS). In classic node sampling, nodes are chosen independently and uniformly at random from the original graph for inclusion in the sampled graph. For a target fraction ϕ of nodes required, each node is simply sampled with a probability of ϕ. Once the nodes are selected, the sampled graph consists of the *induced subgraph* over the selected nodes, *i.e.*, all edges among the sampled nodes are added to form the sampled graph. While node sampling is intuitive and relatively straightforward, the work in [37] shows that it does not accurately capture properties for graphs with power-law degree distributions. Similarly, [24] shows that although node sampling appears to capture nodes of different degrees well, due to its inclusion of all edges for a chosen node set, the original level of connectivity is not likely to be preserved.

Edge sampling (ES). Edge sampling focuses on the selection of edges rather than nodes to populate the sample. Thus, the node selection step in edge sampling algorithm proceeds by just sampling edges, and including both nodes when a particular edge is sampled. The partially induced graph is created just out of the sampled edges, which means no extra edges are added in addition to those chosen during the random edge selection process. Unfortunately, ES fails to preserve many desired graph properties due to the independent sampling of edges. It is however more likely to capture path lengths, due to its bias towards high degree nodes and the inclusion of both end points of selected edges.

Topology-based sampling. Due to the known limitations of NS ([37, 24]) and ES (bias toward high degree nodes), researchers have also considered many other topology-based sampling methods. One example is snowball sampling, which selects nodes using breadth-first search from a randomly selected seed node. Snowball sampling accurately maintains the network connectivity within the snowball, however it suffers from a *boundary bias* in that many peripheral nodes (*i.e.*, those sampled on the last round) will be missing a large number of neighbors [24]. In [26], Leskovec *et al.* propose a Forest Fire Sampling (FFS) method. It starts by picking a node uniformly at random, then'burns' a random fraction of its outgoing links. The process is recursively repeated until no new node is selected or we obtain the sample size. In general, such topology-based sampling approaches perform better than NS and ES.

None of the algorithms discussed above have been explicitly designed to work in a streaming fashion, as the emphasis has been largely on sampling representative subgraphs that matched the properties of the original graph well. In the next section, we discuss our model of graph streams, and show how these standard sampling algorithms could be adapted to work in such a streaming setting. We also propose our new algorithm that outperforms simple streaming variants of these algorithms in the next section.

3. STREAM SAMPLING

We consider an undirected graph $G(V, E)$ with a vertex set $V = \{v_1, v_2, ..., v_N\}$ and edge set $E = \{e_1, e_2, ..., e_M\}$ where N is the number of vertices and M is the number of edges in G. We assume G arrives as a graph stream.

Definition 3.1. *We define a* graph stream *as a sequence of edges* $e_{\pi(1)}, e_{\pi(2)}, ..., e_{\pi(M)}$*, where* π *is any random permutation on* $[M] = \{1, 2, ..., M\}$, $\pi : [M] \to [M]$.

In traditional computational models of graphs, it is difficult to perform random access of the entire graph G at any step, since it is unlikely for large graphs to easily fit in the main memory. A streaming model, in which the graph can only be accessed as a stream of edges, arriving one edge at a time, is therefore more preferable [4].

In a streaming model, as each edge $e \in E$ arrives, the sampling algorithm σ needs to decide whether to include the edge or not as the edge is *streamed* in. The sampling algorithm σ may also maintain state Ψ, and consult the state to determine whether to sample a subsequent edge or not, but the total storage associated with Ψ should be of the order the size of the output sampled graph G_s, *i.e.*, $|\Psi| = O(|G_s|)$. Note that this requirement is potentially larger than the $o(N, t)$ (preferably, $polylog(N, t)$) that streaming algorithms typically require [32]. But, since any algorithm cannot require less space than the output, we relax this requirement in our definition as follows.

Definition 3.2. *We define a* streaming graph sampling algorithm *as any sampling algorithm* σ *that produces a sampled graph* G_s *such that* $|V_s|/|V| = \phi$*, which (1) samples edges of the original graph* $G(V, E)$ *in a sequential order (*i.e.*, not random access) in* one pass*; and, (2) maintains state* Ψ *that is of the order of the size of the sampled graph* G_s*,* i.e.*,* $|\Psi| = O(|G_s|)$.

Now, using the above definition of a streaming graph sampling algorithm, we now present streaming variants of different algorithms discussed in Section 2.

3.1 Streaming Node Sampling

One key problem with traditional node sampling we discussed in Section 2 is that nodes are selected at random. In our stream setting, new nodes arrive into the system only when an edge that contains the new node is added into the system; it is therefore hard to identify which n nodes to select *a priori*. To address this, we essentially use the idea of reservoir sampling [40] and propose the following streaming node sampling variant (outlined in Algorithm 1).

The main idea is to select nodes uniformly at random with the help of a uniform random hash function. Specifically, we keep track of nodes with n smallest hash values in the graph; nodes are only added if their hash values represent the top-n minimum hashes among all nodes seen thus far in the stream. Any edge that has both vertices already in the reservoir is automatically added to the original graph. Since the reservoir is finite, it can happen that a node that arrives much later may have a smaller hash value, in which case it replaces an existing node. All edges incident on that node are then removed from the sampled graph, as there is no chance for those edges to ever get sampled again. Thus, once the reservoir is filled up to n nodes, it will remain at n nodes, but at the same time, it will guarantee sampling from all portions of the stream (not just the front) since the selection is based on the hash value.

Algorithm 1 Streaming NS(Sample Size n, Stream S)

```
▷ $V_s = \emptyset, E_s = \emptyset$
▷ $h$ is fixed uniform random hash function
▷ $t = 1$
for $e_t$ in the graph stream $S$ do
    ▷ $(u, v) = e_t$
    if $u \notin V_s \& \ h(u)$ is top-n min hash then
        $V_s = V_s \cup u$
        Remove all edges incident on replaced node
    end if
    if $v \notin V_s \& \ h(v)$ is top-n min hash then
        $V_s = V_s \cup v$
        Remove all edges incident on replaced node
    end if
    if $u, v \in V_s$ then
        $E_s = E_s \cup e_t$
    end if
    ▷ $t = t + 1$
end for
Output $G_s = (V_s, E_s)$
```

3.2 Streaming Edge Sampling

Streaming edge sampling is a simple variant of the traditional edge sampling. Here, instead of hashing individual nodes, we focus on using hash-based selection of edges (as shown in Algorithm 2). More precisely, if we are interested in obtaining m edges at random from the stream, we can simply keep a reservoir of m edges with the minimum hash value. Thus, if a new edge streams into the system, we check if its hash value is within top-m minimum hash values. If it is not, then we do not select that edge, otherwise we add it to the reservoir while replacing the edge with the previous highest top-m minimum hash value. A similar approach has been proposed by Aggarwal in [2]. However, in his work the goal was to get efficient structural compression of the underlying graph stream rather than getting a representative subgraph that can be used instead of the full graph. One problem with this approach is that our goal is often in terms of sampling a certain number of nodes n. Since we use a reservoir of edges, finding the right m that provides n nodes is really hard. It also keeps varying depending on which edges the algorithm ends up selecting. Note that sampling fraction could also be specified in terms of fraction of edges; the choice of defining it in terms of nodes is somewhat arbitrary in that sense. For our comparison purposes, we ensured that we choose a large enough m such that the number of nodes was much higher than n, but later iteratively pruned out sampled edges with the maximum hash values until the target number of nodes n was reached. While this is not strictly an elegant streaming algorithm, as we shall show in our evaluation, even this extra complexity does not result in producing good graph samples anyway. We include it mainly for comparison purposes.

Algorithm 2 Streaming ES(Sample Size n, Stream S)

```
▷ $V_s = \emptyset, E_s = \emptyset$
▷ $h$ is fixed uniform random hash function
▷ $t = 1$
for $e_t$ in the graph stream $S$ do
    ▷ $(u, v) = e_t$
    if $h(e_t)$ is in top-$m$ min hash then
        $E_s = E_s \cup e_t$
        $V_s = V_s \cup \{u, v\}$
    end if
    Iteratively remove edges in $E_s$ such that $n$ nodes.
    ▷ $t = t + 1$
end for
Output $G_s = (V_s, E_s)$
```

3.3 Streaming Topology-Based Sampling

We also consider a streaming variant of a topology-based sampling algorithm. Specifically, we consider a simple BFS-based algorithm (shown in Algorithm 3) that works as follows. This algorithm essentially implements a simple breadth-first search on a sliding window of w edges in the stream. In many respects, this algorithm is similar to the forest-fire sampling (FFS) algorithm. Just as in FFS, it essentially starts at a random node in the graph and selects an edge to burn (as in FFS parlance) among all edges incident on that node within the sliding window. For every edge burned, let v be the other end of the burned edge. We enqueue v onto a queue Q in order to get a chance to burn its incident edges within the window. For every new streaming edge, the sliding window moves one step, which means the oldest edge in the window is dropped and a new edge is added. (If that oldest edge was sampled, it will still be part of the sampled graph.) If as a result of the sliding window moving one step, the node has no more edges left to burn, then the burning process will dequeue a new node from Q. If the queue is empty, the process jumps to a random node within the sliding window (just as in FFS). This way, it does BFS as much as possible within a sliding window, with random jumps if there is no more edges left to explore. Note that there may be other streaming variants of the sampling algorithm possible; since there are no streaming algorithms in the literature, we chose this as a reasonable approximation for comparison. This algorithm has a similar problem as the edge sampling variant that it is difficult to control the exact number of sampled nodes, and hence some additional pruning needs to be done at the end (as shown in Algorithm 3).

3.4 Partially-Induced Edge Sampling (PIES)

We finally present our main algorithm called PIES that outperforms the above classes of streaming algorithms. In our approach, we mainly exploit the observation that edge sampling is inherently biased towards the selection of nodes with higher degrees, resulting in an *upward bias* in the degree distributions of sampled nodes compared to nodes in the original graph [34]. However, in all sampled subgraphs, degrees are naturally underestimated since only a fraction of neighbors may be selected. This results in a *downward*

Algorithm 3 Streaming BFS(Sample Size n, Stream S,Window Size=$wsize$)

```
▷ $V_s = \emptyset, E_s = \emptyset$
▷ $W = \emptyset$
▷ Add the first $wsize$ edges to $W$
▷ $t = wsize$
▷ Create a queue $Q$
▷ $u$ =random vertex from $W$
for $e_t$ in the graph stream $S$ do
    if $u \notin V_s$ then add $u$ to $V_s$
    if $W.incident\_edges(u) \neq \emptyset$ then
        Sample $e$ from $W.incident\_edges(u)$
        Add $e = (u, v)$ to $E_s$
        Remove $e$ from $W$
        Add $v$ to $V_s$
        enqueue $v$ onto $Q$
    else
        if $Q = \emptyset$ then $u$ =random vertex from $W$
        Else $u = Q.dequeue()$
    end if
    Move the window $W$
    if $|V_s|$ >$n$ then
        Retain $[e] \subset E_s$ such that $[e]$ has $n$ nodes
        Output $G_s = (V_s, E_s)$
    end if
    ▷ $t = t + 1$
end for
Output $G_s = (V_s, E_s)$
```

Algorithm 4 PIES(Sample Size n, Stream S)

```
▷ $V_s = \emptyset, E_s = \emptyset$
▷ $t = 1$
while graph is streaming do
    ▷ $(u, v) = e_t$,
    if $|V_s|$ <$n$ then
        if $u \notin V_s$ then $V_s = V_s \cup \{u\}$
        if $v \notin V_s$ then $V_s = V_s \cup \{v\}$
        $E_s = E_s \cup \{e_t\}$
    else
        ▷ $p_e = \frac{|E_s|}{t}$
        draw $r$ from continuous Uniform(0,1)
        if $r \leq p_e$ then
            draw $i$ and $j$ from discrete Uniform[1,$|V_s|$]
            if $u \notin V_s$ then $V_s = V_s \cup \{u\}$ , drop node $V_s[i]$ with all its incident edges
            if $v \notin V_s$ then $V_s = V_s \cup \{v\}$ , drop node $V_s[j]$ with all its incident edges
        end if
        if $u \in V_s$ AND $v \in V_s$ then $E_s = E_s \cup \{e_t\}$
    end if
    ▷ $t = t + 1$
end while
Output $G_s = (V_s, E_s)$
```

bias, regardless of the actual sampling algorithm used. We also observe that selecting nodes with high degrees results in samples with higher average clustering coefficient and shorter path lengths. It is likely that two interconnected sampled nodes will have the same neighbor if this neighbor is sampled and has an extremely large degree. Additionally, high-degree nodes are usually highly popular in the graph, they serve as good navigators through the graph and the shortest path is usually through those extremely popular ones. Therefore, sampling the high degree nodes can result in connected sampled subgraphs that accurately preserve the properties of the full graph.

However, by sampling edges independently, it is unlikely that the structure of the graph *surrounding* the high degree nodes will be preserved. Thus, we also sample all the edges between any sampled nodes in the graph (graph induction). This helps to recover much of the connectivity around the high degree nodes—offsetting the downward degree bias as well as increasing local clustering in the sampled graph. Graph induction increases the probability that triangles will be formed among the set of sampled nodes, resulting in a higher clustering coefficient and shorter path lengths. The above observations, while simple, makes the sampled graphs approximate the characteristics of the original graph much more accurately, even better than topology-based sampling algorithms.

Unfortunately, full graph induction in a streaming fashion is hard (*i.e.* since it requires at least two passes, when done in the obvious straightforward way). Thus, instead of total induction of the edges between the sampled nodes, we can utilize *partial* induction and combine the edge-based node sampling with the graph induction (as shown in Algorithm 4) into a single step. The partial induction step induces the sample in the forward direction, *i.e.*, adding any edge among a pair of sampled nodes if it occurs after both the two nodes were added to the sample.

PIES aims to maintain a dynamic sample as the graph is streaming utilizing the same reservoir sampling idea we have used before. Specifically, we add the first n records of the stream to a *reservoir* and then the rest of the stream is processed randomly by replacing existing records in the reservoir. PIES will then simply run over the edges in a single pass, adding deterministically the first n nodes of the stream to the sampled graph. Once it achieves the target sample size, then for any streaming edge, it adds the incident nodes to the sample (probabilistically) by replacing other sampled nodes from the node sample set (uniformly at random). At each step, it will also add the edge if its two incident nodes are already in the sampled node set (to produce a partial induction effect).

4. EXPERIMENTAL EVALUATION

In this section, we evaluate the efficacy of the proposed stream sampling algorithms, PIES, NS, ES, and BFS, on several real datasets ranging from about 10,000 - 800,000 nodes, with 30,000 - 6.6 million edges. In our experiments, we consider five real networks: a citation network, a collaboration network, an email communication network, and two online social networks. Table 1 summarizes the characteristics of the (simplified) real networks.

The two data sets called HepPH, and CondMAT, correspond to a citation graph, and collaboration graph respectively, provided by Leskovec *e*t al. [1]. The Facebook data corresponds to Wall communications among users that belong to a New Orleans city [39]. The Twitter dataset contains tweets of users in discussion surrounding the United Nations climate change conference in Dec. 2009. Also, the University email data corresponds to two weeks of email communication we collected from the email logs on Purdue university mailserver(s). We also verify our proposed approach on a large scale graph of 800,000 nodes and 6.6 million edges collected from Flickr network [15].

4.1 Evaluation Measures

We compare four stream sampling methods from different sampling classes. We propose a one-pass implementation of node sampling (NS) and edge sampling (ES) to represent node-based sam-

Dataset	Nodes	Edges	No. CC	Avg. path	Density	Clustering
HEPPH	34,546	420,877	61	4.33	7×10^{-4}	0.146
TWITTER	8,581	27,889	162	4.17	7×10^{-4}	0.061
FACEBOOK (NO)	46,952	183,412	842	5.6	2×10^{-4}	0.085
CONDMAT	23,133	93,439	567	5.35	4×10^{-4}	0.264
EMAIL-PU UNIV	214,893	1,270,285	24	3.91	5.5×10^{-5}	0.0018
FLICKR	820,878	6,625,280	1	5.01	1.9×10^{-5}	0.116

Table 1: Characteristics of Network Datasets

pling and edge-based sampling classes respectively. We also propose a one-pass breadth first sampling (BFS) to represent the topology-based sampling class. We implement BFS on a sliding window of 100 edges of the stream. Our evaluation is primarily along four main properties—degree, path length, clustering coefficient, and size of weakly connected components. We conjecture these four properties capture both local and global characteristics of the graph. We measure the performance of a sampling algorithm by how well the sampled subgraphs preserve the probability density function (PDF) and complementary cumulative distribution function (CCDF) of each of these four properties. Unlike other measure based on aggregate statistics (*e.g.*, average degree, density, reciprocity), these four measures represent the distribution of properties across the nodes and edges in the sample, which facilitates detailed comparison and evaluation of sample representativeness.

In addition to visually comparing the similarity of the distributions on the sampled subgraphs to those of the original graphs, we also compute two statistics to compare the distributions quantitatively across different sampling fractions. First, we use the Kolmogorov-Smirnov (KS) statistic to assess the distance between two cumulative distribution functions (CDF). The KS-statistic is a widely used measure of the agreement between two distributions; the authors of [26] also have used the KS distance to illustrate the accuracy of FFS samples in the past. It is computed as the maximum absolute distance between the two distributions, where x represents the range of the random variable and F_1 and F_2 represent two CDFs. In this work, F_1 represents the true distribution of the full graph and F_2 represents the approximation of F_1 calculated from the sampled subgraph.

$$KS(F_1, F_2) = max_x |F_1(x) - F_2(x)| \qquad (1)$$

We also used another statistical measure for evaluation called skew divergence, in order to measure the Kullback-Leibler (KL) divergence between two distributions that do not have the same continuous support over the range of values [23]. The results of skew divergence are similar to the KS statistic results, therefore we omitted them to save space.

4.2 Results

In our experiments, we focus on obtaining a sample between 5–40% ($\phi = 0.05$ to 0.40) of the full graph. For each sample fraction, we experiment with ten different runs, and in each run, we generate a sample from a new random seed. It is unlikely to assume a certain order of edges in the stream because usually social communication among users can happen in any arbitrary order. To simulate this aspect we randomly permute the edges in the graph in each run.

KS-statistic. We compute the average of each of these measures across the five datasets and ten runs for each metric. Figures 1(a)–1(d) show the average KS-statistic for degree, path length, clustering coefficient and size of connected components, respectively. We observe that PIES outperforms BFS, NS, and ES for degree, path length, and clustering coefficient. NS comes in the second rank after PIES for the aforementioned measures. Both BFS and ES outperform PIES and NS on the size of connected components, but they do not perform well on the other measures. Overall, all sampling algorithms that include an induced graph step (PIES and NS) in their process perform well for the cases of degree, path length and clustering coefficient as they capture more edges between the sampled nodes.

Distributions. We plot the distributions of the three metrics in Figure 4 for Facebook (a-c), and Email Purdue university (d-f) at 20% sampling fraction. We picked the 20% sampling fraction as a reasonable sample size to show the difference between the distributions of different sampling algorithms. However, other sampling proportions show similar relative behavior among the algorithms.

Figures 2(a) and 2(d) show the degree distribution for the two networks. From the figures, we can observe that NS under-estimates the degree of the nodes, resulting in a large fraction of zero-degree (low-degree) nodes in its sample across the two networks. Similarly, BFS and ES also capture a large fraction of low-degree nodes.

Figures 2(b) and 2(e) show the path length distribution for the two networks. we observe NS samples have a high fraction of long path lengths compared to PIES since it samples low-degree nodes more than high-degree nodes.

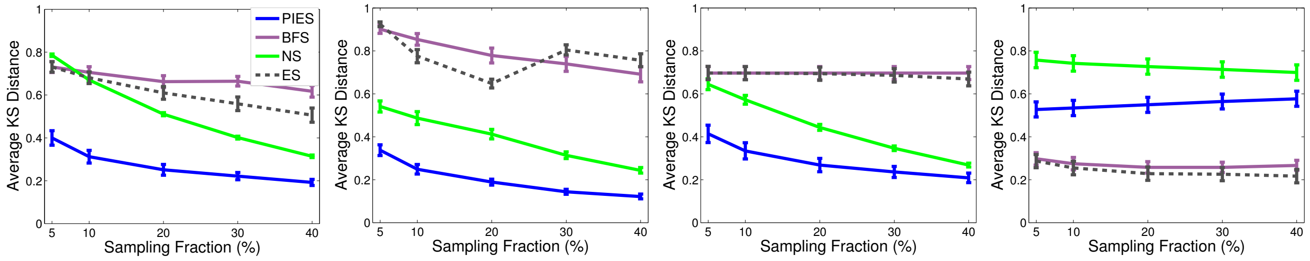
Figures 2(c) and 2(f) show the clustering coefficient distributions. Across the two networks, NS, ES and BFS produce unclustered samples. PIES performs well for both Email and Facebook networks, however, it performs similar to NS on the HepPH network.

Overall, PIES is the closest to preserving the three distributions compared to other methods. This is due to the fact that PIES samples high degree nodes with a larger probability than NS. Similar to PIES, both BFS and ES select high degree nodes with a probability higher than NS. However, PIES outperforms BFS and ES since it adds extra edges between the sampled nodes (i.e. through partial induction in the forward direction).

We omitted the plots for the size of weakly connected components due to the limited space, however, ES and PIES outperformed the other methods.

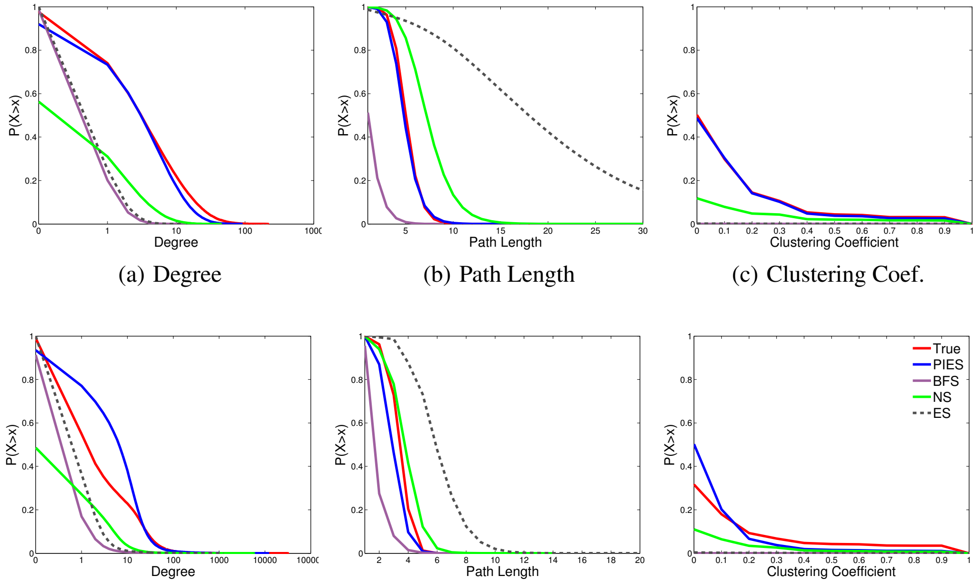
In addition to analyzing the KS statistic as an average on all networks, we also analyze the performance of PIES for each network in Figure 3 (average over all graph properties), sorting the networks in increasing order from left to right in terms of their density and clustering. The results indicate that PIES performs better in datasets that are less dense/clustered. This is an interesting result that shows PIES will be more suitable to sample rapidly changing graph streams that are more likely to have a lower density over time.

Evaluation on different points of the stream. Further, Figures 4(a), and 4(b) show the KS statistics (average over all graph properties) of the different algorithms at different points in the stream while it is progressing. PIES performs better than NS, BFS, and ES on Facebook. However, PIES performs slightly better than other meth-



(a) Degree (b) Path length (c) Clustering Coefficient (d) Size of Weakly Conn. Comp.

Figure 1: Average KS Distance across 5 datasets.



(a) Degree (b) Path Length (c) Clustering Coef.

(d) Degree (e) Path Length (f) Clustering Coef.

Figure 2: Distributions at 20% sampling fraction: (a-c)Facebook New Orleans , (d-f)Email Purdue University.

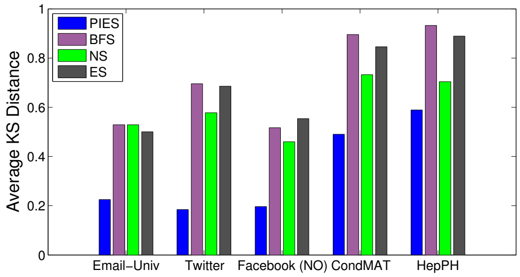
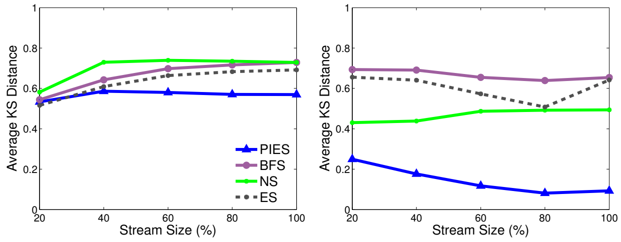


Figure 3: Average KS Statistics for different networks (sorted in increasing order of clustering/density from left to right).



(a) HepPH (b) Facebook

Figure 4: Average KS Statistics at different points of the stream

ods on HepPh. This also illustrates that PIES can maintain a consistently good random sample at different lengths of the stream.

Back-in Time Goal. Leskovec *et al.* proposed the back-in time sampling goal [26] which corresponds to traveling back in time and capturing properties of the past versions of G at sizes $n' < n$. In this experiment, we investigate the question whether we can sample in a manner that allows us to match what the stream looked like in the past. This can help in studying the stationarity properties of the graph stream as it evolves over the time. Figure 5 shows the average KS statistics (average over all graph properties) of the different algorithms when the goal is to approximate the graph stream back-in time when it was only 20% the size of the full stream. We again observe that PIES performs better than the other algorithms. We show the results only for Facebook and HepPh networks, however the same conclusions apply for the other datasets.

Sampling from Very Large graphs. While sampling from small graphs (*i.e.* with thousands of nodes/edges) is important for many applications, it is unrealistic for many other applications that deal with very large graphs with hundreds of thousands of nodes/edges. These large graphs are typically too big to fit into memory and

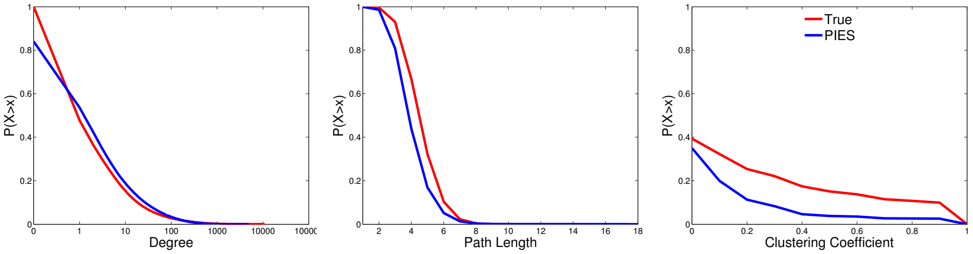


Figure 6: Distributions at 30% sampling fraction for Flickr Network

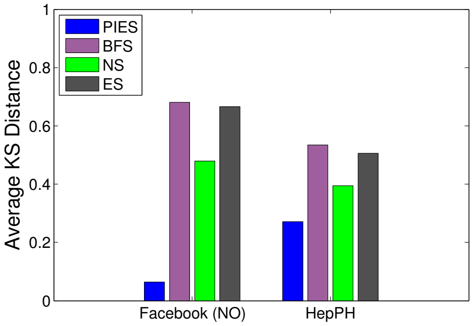


Figure 5: Average KS Statistics when the goal is to match the graph back-in time at 20% of the stream

therefore they are hard to process with existing sampling methods. Therefore, we also verified our proposed algorithm PIES on large scale graphs with 800,000 nodes and 6,6 million edges collected from Flickr network. As shown in figure 6, PIES sampled graphs are close to the properties of the larger Flickr network using only *a single* pass on the edges [1].

Comparison with non-streaming algorithms. Our goal is to obtain a representative sample from a stream that is either evolving over the time or too large to fit into memory. Here we compare PIES to other non-streaming sampling algorithms. We compare to Forest Fire Sampling (FFS) [26] and fully-induced edge sampling (ES-i). In the case of FFS, we use $p_f = 0.7$ as in [26]. In ES-i, we first sample the edges with ES then we add all the edges among the sampled nodes in a second pass (full induction).

Figure 7 shows the average KS statistic (average over all graph properties) for the five networks. Overall, PIES performs better than both ES-i and FFS. However, ES-i performs better for HepPh and ConMAT. This illustrates the effect of full induction versus partial induction for more dense networks. Since ES-i gets the chance to add more edges among the sampled nodes, it outperforms PIES on graphs with higher density/clustering. However, PIES performs better for the less dense, and clustered graphs.

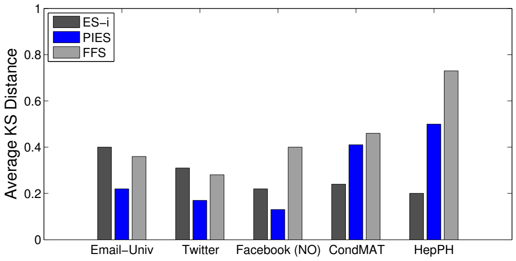


Figure 7: Average KS Statistics for different networks (sorted in increasing order of clustering/density from left to right).

5. RELATED WORK

Sampling from Graphs. The problem of sampling graphs has been of interest in many different fields of research. The work in [24, 42, 37] studies the statistical properties of samples from complex networks produced by traditional sampling algorithms such as node sampling, edge sampling and random-walk based sampling and discusses the biases in estimates of graph metrics due to sampling. The work in [29] also discusses the connections between specific biases and various measures of structural representativeness. In addition, there have been a number of sampling algorithms in other communities such as in peer-to-peer networks [38, 14]. Internet modeling research community [20] and the WWW information retrieval community has focussed on random walk based sampling algorithms like PageRank [33, 18]. There is also some work that highlights the different aspects of the sampling problem. Examples include [19, 8, 5]

In social networks research, the recent work in [34] uses random walks to estimate node properties in G (e.g., degree distributions in online social networks). These different sampling algorithms focused on estimating either the local or global properties of the original graph, but *not* to sample a representative subgraph of the original graph, which is our goal. The work in [28] studied the problem of sampling a subgraph representative of the graph community structure by sampling the nodes that maximize the expansion.

Due to the popularity of online social networks such as Facebook and Twitter, there has been a lot of work [31, 27, 25, 21, 7, 11] studying the growth and evolution of these networks. While most of them have been on static graphs, recent works [41, 39] have started focusing on interactions in social networks. There is also work on decentralized search and crawling [10, 13, 22], however, in our work we focus on sampling from graphs that are naturally evolving as a stream of edges. In the literature, the most closely related efforts are that of Leskovec *et al.* in [26] and Hubler *et al.* in [17]. But, as we mentioned before, our work is different as we focus on the novel problem of sampling from graphs that are naturally evolving as a stream of edges (graph streams).

Impact of Sampling on Other applications. Recently, some research has also focused on how the different sampling methods impact the performance of applications overlaid on the networks. One

[1] Note that for the Flickr data experiments, we compare PIES to the true distribution only, since the other baseline methods are inefficient to run for very large graphs and they don't match the graph properties well on smaller sampling sizes.

such study investigated the impact of sampling designs on the discovery of the information diffusion process [12]. Another study investigated the impact of the choice of the sampling design on the performance of relational classification algorithms [6].

Data and Graph Streams. Although significant work has been proposed to solve the problem of graph sampling, to our knowledge, there is no prior research on sampling from graph streams to obtain a representative subgraph. However, several research works [36, 9, 16] studied graph streaming algorithms for counting triangles, degree sequences, and estimating page ranks. The main contributions of these works are to use a small amount of memory (sublinear space) and few passes to perform computations on large graphs streams. In database research, some research studied data stream management systems. For example, the work in [30] studied the problem of computing frequency counts in data streams, and the work in [3] studied the problem of sampling from data stream of database queries.

6. CONCLUSIONS

Much of the past efforts on sampling networks have assumed that the sampling algorithm can access the full graph in order to decide which nodes/edges to select. However, many large-scale network datasets are constructed from a graph *stream* consisting of micro-communications among users (*e.g.* wall posts, tweets, emails). In this work, we have formulated the problem of sampling *representative* subgraphs from such large graph streams. We proposed a novel sampling algorithm, PIES, that is based on combining edge sampling with partial induction. Our approach is not only simple and efficient, it is also amenable to a streaming implementation. Furthermore, our empirical results show that PIES significantly outperforms other sampling algorithms, both streaming and non-streaming, across a range of real-world network datasets. In future work, we aim to study the theoretical properties of graph stream sampling in particular our proposed method.

7. ACKNOWLEDGEMENTS

This research is supported by ARO, NSF under contract number(s) W911NF-08-1-0238, IIS-1017898, IIS-0916686, IIS-1149789. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements either expressed or implied, of ARO, NSF, or the U.S. Government.

8. REFERENCES

[1] Stanford large network dataset collection. `http://snap.stanford.edu/data/index.html`.

[2] C. Aggarwal, Y. Zhao, and P. Yu. Outlier detection in graph streams. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 399–409. IEEE, 2011.

[3] C. C. Aggarwal. On Biased Reservoir Sampling in the Presence of Stream Evolution. In *Proceedings of the 32nd International Conference on Very Large Data Bases.*, VLDB '06, pages 607–618, 2006.

[4] C. C. Aggarwal and H. Wang, editors. *Managing and Mining Graph Data*, volume 40 of *Advances in Database Systems*. Springer, 2010.

[5] N. K. Ahmed, J. Neville, and R. Kompella. Reconsidering the foundations of network sampling. *WIN'10*, 2010.

[6] N. K. Ahmed, J. Neville, and R. Kompella. Network sampling designs for relational classification. In *Sixth International AAAI Conference on Weblogs and Social Media*, 2012.

[7] Y. Ahn, S. Han, H. Kwak, S. Moon, and H. Jeong. Analysis of topological characteristics of huge online social networking services. In *WWW*, pages 835–844, 2007.

[8] K. Avrachenkov, B. Ribeiro, and D. Towsley. Improving random walk estimation accuracy with uniform restarts. *Algorithms and Models for the Web-Graph*, pages 98–109, 2010.

[9] Z. Bar-Yossef, R. Kumar, and D. Sivakumar. Reductions in streaming algorithms with an application to counting triangles in graphs. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms*, SODA '02, pages 623–632, 2002.

[10] E. Baykan, M. Henzinger, S. Keller, S. De Castelberg, and M. Kinzler. A comparison of techniques for sampling web pages. *Arxiv preprint arXiv:0902.1604*, 2009.

[11] H. Chun, H. Kwak, Y. Eom, Y. Ahn, S. Moon, and H. Jeong. Comparison of online social relations in volume vs interaction: a case study of cyworld. In *ACM/USENIX IMC*, pages 57–70, 2008.

[12] M. De Choudhury, Y. Lin, H. Sundaram, K. Candan, L. Xie, and A. Kelliher. How does the data sampling strategy impact the discovery of information diffusion in social media. In *Proceedings of the 4th International AAAI Conference on Weblogs and Social Media*, pages 34–41, 2010.

[13] M. Gjoka, M. Kurant, C. Butts, and A. Markopoulou. Walking in facebook: A case study of unbiased sampling of osns. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9. Ieee, 2010.

[14] C. Gkantsidis, M. Mihail, and A. Saberi. Random walks in peer-to-peer networks. In *IEEE INFOCOM*, 2004.

[15] D. F. Gleich. Graph of flickr photo-sharing social network crawled in may 2006, Feb 2012.

[16] S. Guha, A. McGregor, and S. Venkatasubramanian. Streaming and sublinear approximation of entropy and information distances . In *ACM-SIAM Symposium on Discrete Algorithms*, SODA '06, pages 733–742, 2006.

[17] C. Hubler, H.-P. Kriegel, K. M. Borgwardt, and Z. Ghahramani. Metropolis algorithms for representative subgraph sampling. In *ICDM*, 2008.

[18] J. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.

[19] E. Kolaczyk. Statistical analysis of network data, volume 69 of springer series in statistics, 2009.

[20] V. Krishnamurthy, M. Faloutsos, M. Chrobak, J. Cui, L. Lao, and A. Percus. Sampling large Internet topologies for simulation purposes. *Computer Networks*, 51(15):4284–4302, 2007.

[21] R. Kumar, J. Novak, and A. Tomkins. Structure and evolution of online social networks. In *SIGKDD*, pages 611–617, 2006.

[22] M. Kurant, A. Markopoulou, and P. Thiran. Towards unbiased bfs sampling. *Selected Areas in Communications, IEEE Journal on*, 29(9):1799–1809, 2011.

[23] L. Lee. On the effectiveness of the skew divergence for statistical language analysis. In *Artificial Intelligence and Statistics*, 2001.

[24] S. Lee, P. Kim, and H. Jeong. Statistical properties of sampled networks. *Physical Review E*, 73:016102, 2006.

[25] J. Leskovec, L. Backstrom, R. Kumar, and A. Tomkins. Microscopic evolution of social networks. In *SIGKDD*, 2008.

[26] J. Leskovec and C. Faloutsos. Sampling from large graphs. In *SIGKDD*, pages 631–636, 2006.

[27] J. Leskovec and E. Horvitz. Worldwide Buzz: Planetary-Scale Views on an Instant-Messaging Network. In *WWW*, 2008.

[28] A. S. Maiya and T. Y. Berger-Wolf. Sampling Community Structure. In *WWW*, 2010.

[29] A. S. Maiya and T. Y. Berger-Wolf. Benefits of bias: Towards better characterization of network sampling. In *SIGKDD*, 2011.

[30] G. S. Manku and R. Motwani. Approximate Frequency Counts over Data Streams. In *Proceedings of the 28th International Conference on Very Large Data Bases.*, VLDB '02, pages 346–357, 2002.

[31] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and analysis of online social networks. In *ACM/USENIX IMC*, 2007.

[32] S. Muthukrishnan. Data streams: algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(2), Aug. 2005.

[33] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. 1998.

[34] B. Ribeiro and D. Towsley. Estimating and sampling graphs with multidimensional random walks. In *ACM SIGCOMM Internet Measurement Conference*, Nov. 2010.

[35] F. P. Room. Facebook statistics. `http://www.facebook.com/press/info.php?statistics`.

[36] A. D. Sarma, S. Gollapudi, and R. Panigrahy. Estimating PageRank on Graph Streams. In *PODS*, 2008.

[37] M. Stumpf, C. Wiuf, and R. May. Subnets of scale-free networks are not scale-free: Sampling properties of networks. *Proceedings of the National Academy of Sciences*, 102(12):4221–4224, 2005.

[38] D. Stutzbach, R. Rejaie, N. Duffield, S. Sen, and W. Willinger. On unbiased sampling for unstructured peer-to-peer networks. In *IMC*, pages 27–40, 2006.

[39] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi. On the evolution of user interaction in facebook. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Social Networks (WOSN'09)*, August 2009.

[40] J. Vitter. Random sampling with a reservoir. *ACM Trans. Math. Softw.*, 11, 1985.

[41] C. Wilson, B. Boe, A. Sala, K. P. Puttaswamy, and B. Y. Zhao. User interactions in social networks and their implications. In *EuroSys*, pages 205–218, 2009.

[42] S. Yoon, S. Lee, S.-H. Yook, and Y. Kim. Statistical properties of sampled networks by random walks. *Phys. Rev. E*, 75(4):046114, Apr 2007.