ComPAS: Community Preserving Sampling for Streaming Graphs

[1]Sandipan Sikdar, [2]Tanmoy Chakraborty, [3]Soumya Sarkar,
[4]Niloy Ganguly, [5]Animesh Mukherjee
[1,3,4,5]Indian Institute of Technology Kharagpur, India;
[2]University of Maryland, College Park
{[1]sandipansikdar,[3]soumya015,[4]niloy,[5]animeshm}@cse.iitkgp.ernet.in,
[2]tanchak@umiacs.umd.edu

ABSTRACT

In the era of big data, graph sampling is inevitable in many settings. Existing sampling methods are mostly designed for static graphs, and aim to preserve *major* structural properties of the original graph (such as degree distribution, clustering coefficient etc.) in the sample. We posit that for any sampling method it is impossible to produce universal representative which can preserve *all* the properties of the original graph; rather sampling should be application specific (such as preserving hubs for information diffusion). Here we consider *community detection* as an application and propose `ComPAS`, a novel sampling strategy that unlike previous methods, is not only designed for *streaming graphs* (which is more realistic representation of a graph) but also *preserves the community structure* of the original graph in the sample. `ComPAS` interweaves graph sampling and community detection in such a way that each gets benefits from the other to produce a community-preserved sample as well as its associated community structure (as a bi-product).

Empirical results on both synthetic and different real-world graphs show that `ComPAS` is the best to preserve the underlying community structure and quite competitive in maintaining general graph properties with average performance reaching 85.5% of the most informed algorithm on static graphs. Finally, we present additional benefits of `ComPAS` through two applications – selection of right community detection algorithm for a particular graph and selection of training set for online learning. For both the applications `ComPAS` performs almost as good as the most informed baseline for static graphs.

1. INTRODUCTION

Nowadays graphs are so large that their analysis in entirety can be intractable and impractical. How then one should proceed in analyzing and mining these graphs? Traditional approaches include designing more efficient algorithms or leveraging computing power through parallelization or distributed computing. Unfortunately, these existing methods are not always easily available as an option. Another approach that has received recent attention is the technique of *sampling* [23].

Although data sampling is exhaustively studied in statistics [8, 16], sampling from graphs has got only limited attention [14, 24, 31, 32]. Moreover, when it comes to the case of sampling from *streaming graphs* (where edges arrive in discrete time intervals) there is hardly any work beside [2, 4]. Existing graph sampling methods are mostly designed for *static graphs* and aim at preserving general structural properties (such as degree distribution, clustering coefficient etc.) of the original graph in the sample. However we posit that it is impossible for any sampling method to produce a universal representation that can preserve *all* sorts of graph properties of the original graph; rather graph sampling should be *application specific*. For instance, sampling method designed for information diffusion should preserve the hubs (high-degree nodes) in the sample; whereas sampling for outbreak detection (such as disease outbreak) should preserve the nodes with high local clustering coefficient.

In this paper, we propose a novel sampling algorithm that preserves the original *community structure*[1]. A community in a graph is a cluster of nodes more densely connected among themselves than to others [36]. Identifying community structure is important, as they represent the *mesoscopic view* of the graph and often correspond to real social groups, functional groups, or demographic similarity etc. [13]. The ability to easily construct a sample consisting of members from diverse communities has several important applications. For instance, in marketing, surveys often seek to construct stratified samples that collectively represent the diversity of the population [20]. Many popular community detection algorithms considered to be accurate are also computationally expensive [7, 9]. Representative graph sampling, then, provides a potential solution for inferring and approximating global, latent properties such as these in large graphs. By sampling a representative subgraph, analysis can be performed on the sample instead of the larger graph. Results could, then, be generalized to the larger population, which, in this case, is the original graph. However, if one still intends to run an accurate and computationally-expensive community detection algorithm on large graphs and is confused which one algorithm to choose, she can run several potential algorithms on the sample graph and choose one that performs best. Then that algorithm can be used to detect communities from the original large graph.

Our contributions: In this paper, we propose `ComPAS`, a novel sampling algorithm on streaming graph (most realistic graph representation [2, 4]) that is capable of representing and inferring community structure in the original graph. `ComPAS` systematically interweaves graph sampling and community detection so that one gets benefits from the other to produce a more representative sample. In particular, our contributions are three-fold:

[1]In this paper, we consider disjoint community structure.

Table 1: Important notations used in this paper.

Notation	Description
S	Graph stream $\{e_1, e_2, \dots\}$
n	Required sample size (in terms of nodes)
G_s	$G_s = (V_s, E_s)$, final graph sample
C_s	Community structure of G_s
α	Initial fraction of nodes inserted
$Algo$	Algorithm used to detect initial community structure
$N(x)$	Neighbor of x
$P(x)$	Parent of x
$\mathcal{H}$	Buffer consisting of $\mathcal{H}_c$ and $\mathcal{H}_p$
$\mathcal{H}_c$	Dictionary tracking number of times each node is encountered
$\mathcal{H}_p$	Dictionary storing the recent parent of each node
n_d	Size of $\mathcal{H}$
D_s	Sum of degree of all nodes inside set s
$C(v)$	Community of node v

- To our knowledge `ComPAS` is the first *community-preserving* sampling method for *streaming graphs*. Along with the sample, `ComPAS` also produces the community structure of the sample.

- Empirical evidences on synthetic graph and different real-world graph demonstrate that the sample generated by `ComPAS` is not only the most representative to preserve the community structure, but also is quite competitive in reproducing the other general graph properties. The average performance of our algorithm reaches 85.5% of the most informed algorithm (GA [35]) on static graphs.

- We show additional benefits of `ComPAS` through two applications – (i) selection of right community detection algorithm for large graphs, and (ii) selection of (limited) training set for online learning. We obtain a performance that is within 95.6% and 90.5% of the most informed algorithm (i.e., GA) available for static graphs for first and second applications respectively.

2. RELATED WORK

The problem of sampling from a population has long been studied in social sciences [10, 11], such as snowball sampling [15], respondent-driven sampling [17, 12] etc. Most of the relevant works on sampling deal with estimating global properties of the population (see a survey in [21]).

With the advent of large-scale graph data, sampling problem has received a renewed interest with an initial attempt in [23]. Although several sampling algorithms have been proposed since then [32, 31], it is very difficult to obtain a universally representative sample that preserves *all* the properties of the original graph [3, 24, 25].

Most sampling algorithms work on static graphs, meaning that the entire graph should be available in advance. With increasing interest in sampling more practical social graph (which are mostly dynamic in nature), the focus has recently shifted towards sampling from streaming graphs [18]. A streaming graph is considered to be a stream of incoming edges (see Figure 1). [2] proposed a streaming edge sampling (SE) algorithm for outlier detection. [4] proposed streaming node sampling (SN), streaming BFS (Breadth First Search, SBFS) and Partially Induced Edge Sampling (PIES) algorithms. All of these algorithms aim to preserve the general graph properties of the original graph in the sample.

In this work we propose a sampling algorithm `ComPAS` for streaming graphs that is primarily designed to obtain a sample which is most representative of the original graph in terms of the underlying community structure. [35] claimed that their proposed Green Algorithm (GA) can generate sample from a *static graph* that preserves the community structure (as we attempt here), although they did not produce the community structure explicitly. To the best our knowledge, *ours is the first attempt toward obtaining a sample that preserves the community structure of the streaming graph.* Further, our algorithm is quite competitive in retaining the other graph properties (such as degree distribution, clustering coefficient) in the sample (see Section 6.2).

3. PROBLEM DEFINITION

We consider a graph stream S represented by a set of edges $e_1, e_2, ...$ with each edge e_i arriving at i^{th} (discrete) time step. A graph G at time t is the aggregate of all the edges arriving till time t. V represents the set of unique nodes present in the graph G. The community structure of the graph G is represented by C. We consider G to be both unweighted and undirected (see Table 1 for the notations).

DEFINITION 1. *Given a streaming graph G and sample size n in terms of the number of nodes, our objective is to obtain a sample graph G_s such that C, the underlying community structure of G is highly preserved in G_s (i.e., $C \sim C_s$ where C_s is the community structure of G_s).*

Algorithm 1: `ComPAS`: A **Com**munity **P**reserving Sampling **A**lgorithm for **S**treaming Graph

Data: S: Graph stream, n: Sample size, α: Initial fraction of nodes inserted, β: Edge density threshold, n_d: size of the buffer, $Algo$: a community detection algorithm
Result: Sampled Subgraph $G_s(V_s, E_s)$, C_s

Initialize G_s: $V_s = \phi$, $E_s = \phi$
Crate an empty buffer $\mathcal{H}$ of size n_d
Initialize buffer $\mathcal{H}$: $\mathcal{H}_c = \phi$, $\mathcal{H}_p = \phi$
$flag = 1$, $t = 0$
for e_t *in the graph stream* S **do**
  $e_t = \{u, v\}$
  if $\frac{|V_s|}{n} < \alpha \wedge e_t \notin E_t$ **then**
    $V_s = V_s \cup u \cup v$
    $E_s = E_s \cup e_t$
    Continue;
  else if *flag==1* **then**
    Run $Algo$ on G_s and detect community structure C_s
    $flag$=0
  else if $u, v \in V_s$ **then**
    $V_s, E_s, C_s = BothinSample(u, v, e_t, V_s, E_s, C_s)$
  else if $u \in V_s \wedge v \notin V_s \wedge v \notin \mathcal{H}$ **then**
    $V_s, E_s, C_s, \mathcal{H} = OneinSampleOneNew(u, v, e_t, V_s, E_s, \mathcal{H}, C_s)$
  else if $u \in V_s \wedge v \notin V_s \wedge v \in \mathcal{H}$ **then**
    $V_s, E_s, C_s, \mathcal{H} = OneinSampleOneinBuffer(u, v, e_t, V_s, E_s, \mathcal{H}, C_s)$
  else if $u \notin V_s \wedge u \in \mathcal{H} \wedge v \notin V_s \wedge v \notin \mathcal{H}$ **then**
    $V_s, E_s, C_s, \mathcal{H} = OneinBufferOneNew(u, v, e_t, V_s, E_s, \mathcal{H}, C_s)$
  else if $u, v \notin V_s \wedge u, v \notin \mathcal{H}$ **then**
    $V_s, E_s, C_s, \mathcal{H} = BothNew(u, v, e_t, V_s, E_s, \mathcal{H}, C_s)$
  else if $u, v \notin V_s \wedge u, v \in \mathcal{H}$ **then**
    $\mathcal{H} = BothinBuffer(u, v, \mathcal{H})$
  $t = t + 1$
return G_s, C_s

4. PROPOSED ALGORITHM: COMPAS

Here we propose `ComPAS`, a **Com**munity **P**reserving sampling **A**lgorithm for **S**treaming graphs. `ComPAS` aims at sampling a streaming graph in such a way that its underlying

community structure is preserved in the sample (a pseudo-code and a toy example are presented in Algorithm 1 and Figure 1 respectively). To start with, ComPAS keeps on adding streaming edges (nodes) into the sample G_s as long as a certain fraction of nodes α is inserted (lines 7-10). This in turn provides an initial knowledge about the graph structure. Once the threshold is reached, a pre-selected community detection algorithm *Algo* is run on G_s to detect the initial community structure (line 12). After that, it interweaves both graph sampling and community detection in such a way that each task gets benefits from the other. Once an edge e_t is taken from the stream, ComPAS judiciously inserts e_t into G_s with the help of a buffer $\mathcal{H}$ which is composed of $\mathcal{H}_c$ and $\mathcal{H}_p$. $\mathcal{H}_c$ counts the "number of hits" of a node (i.e., number of times a node is encountered till that time)[2], and $\mathcal{H}_p$ keeps track of the current parent of a node (i.e., node with which it arrived last). A streaming edge $e_t = \{u, v\}$ is first inserted into $\mathcal{H}$, and depending upon the current position of u and v (whether in the buffer or in the sample), their counts in $\mathcal{H}_c$, their current parents in $\mathcal{H}_p$, and the current community structure C_s of G_s, a decision that which node/edge is to be inserted into G_s is taken. One of the six different submodules is invoked to systematically handle this decision. Throughout the iterations, ComPAS maximizes *modularity* [27], a well-studied objective function for community detection defined below:

$$Q(G(V,E),C) = \sum_{c \in C}\left(\frac{m_c}{M} - \frac{D_c^2}{4M^2}\right) \quad (1)$$

where C is the community structure of G, m_c is the total number of edges inside c, D_c is the sum of degree of all the nodes inside a community $c \in C$, and $M = |E|$ is the total number of edges G. In this section, we individually present each submodule of ComPAS in details.

Once the initial threshold α is reached, a community finding algorithm *Algo* is used to detect the initial community structure of G_s (denoted as C_s) (line 12). Then in every occurrence of a new edge $e_t = \{u, v\}$, it is not allowed to enter into the sample immediately; instead depending upon the current position of u and v different subroutines are invoked to place the edge (i.e., two end nodes) into the buffer. This in turn may remove an existing node out of the buffer and connect it with its parent in G_s. C_s is adjusted accordingly. In this section, we describe different subroutines in details.

(i) Both u and v are present in the sample: When both u, v are in V_s, *BothinSample*() (see Function 1) is called from line 15 of Algorithm 1. We further divide this case into two subcases: e_t is an intra-community edge (totally inside a single community) or an inter-community edge (connecting two communities $C(u)$ and $C(v)$). In the former case (edge $\{b, d\}$ in Figure 1), addition of e_t will strengthen the internal community structure according to Proposition 1[3]. We also know from Proposition 2 that adding an intra-community edge should not split the current community. Therefore we leave C_s in its current form without any modification.

In case of e_t connecting two different communities (edge $\{b, f\}$ in Figure 1), several possibilities may arise. u (or v) may leave its current community and join in other community. Additionally, if the community membership of u (or v) is changed, it can also pull out its neighbors to join

[2]In streaming graph, an edge might appear multiple times.
[3]Detailed proofs of all the propositions can be found in [1].

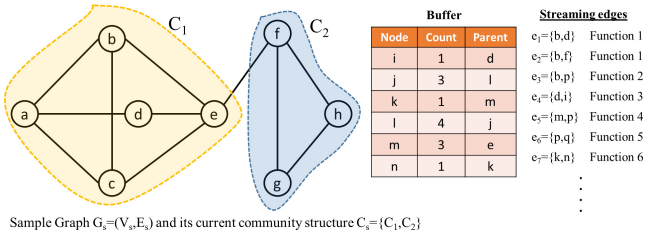

Figure 1: Toy example depicting various conditions handled by ComPAS when a streaming edge arrives.

with it, and some of the neighbors might eventually want to change their memberships as well. According to Proposition 3, we know that if u (or v) ever changes its community membership, $C(v)$ (or $C(u)$) would be the best new community for it. But how do we quickly decide it? Here we provide a criteria to check the change in membership for u and v in Proposition 4. If both $\Delta Q(u, C(u), C(v))$ and $\Delta Q(v, C(v), C(u))$ (where $\Delta Q(u, C(u), C(v))$ indicates the change in modularity after assigning u from $C(u)$ to $C(v)$) fail to satisfy the criteria (see Corollary 1), we can retain the current community structure. Otherwise, we move u (or v) to $C(v)$ (or $C(u)$) and consequently we let its neighbors decide their best move in the similar way.

PROPOSITION 1. *For a community $c \in C$, if $D_c \leq M - 1$ (where $M = |E|$) then addition of an edge within c will increase its modularity.*

PROOF. From Equation 1, we see the contribution of individual community $c \in C$ in modularity as: $Q_c = \frac{m_c}{M} - \frac{D_c^2}{4M^2}$.

Addition of a new edge within c, the c's contribution of modularity becomes:

$$Q'_c = \frac{m_c + 1}{M + 1} - \frac{(D_c + 2)^2}{4(M+1)^2}$$

So the increase in modularity is $\Delta Q_c = Q'_c - Q_c$,

$$\begin{aligned}\Delta Q_c =& \frac{4M^2 - 4m_cM^2 - 4D_cM^2 - 4m_cM + 2D_c^2M + D_c^2}{4(M+1)^2M^2} \\ \geq& \frac{(2M^2 - 2D_cM - D_c)(2M - D_c)}{4(M+1)^2M^2} \\ \geq& 0\end{aligned}$$

The equality holds if $D_c \leq M-1$. This thus implies $(2M^2 - 2D_cM - D_c) \geq 0$. This proves the proposition. □

PROPOSITION 2. *For a community $c \in C$, addition of any intra-community edge into c should not split it into smaller communities.*

PROOF. We will prove this proposition by contradiction. Assume that once a new intra-community edge is added into c, it gets split into k small modules, namely X_1, X_2, $\cdot$,X_k. Let D_{X_i} and e_{ij} be the total degree of nodes inside X_i and number of edges connecting X_i and X_j respectively.

Before adding the edge, we have $Q_c \geq \sum_{i=1}^{k} Q_{X_i}$ (where Q_c is the total modularity of community c), because otherwise all X_is can be split earlier, which is not in this case. This implies that: $\frac{m_c}{M} - \frac{D_c^2}{4M^2} > \sum_{i=1}^{k}(\frac{m_{X_i}}{M} - \frac{D_{X_i}^2}{4M^2})$. Since $X_1, X_2, \cdot, X_k$ are all disjoint modules of c, $D_c = \sum_{i=1}^{k} D_{X_i}$ and $m_c = \sum_{i=1}^{k} m_{X_i} + \sum_{i<j} e_{ij}$. This further implies that: $\sum_{i<j} e_{ij} > \frac{\sum_{i<j} D_{X_i} D_{X_j}}{2M}$.

Without loss of generality, let us assume that the new edge is added inside X_1. Since we assume that after adding the

new edge into c, it gets split into k small modules, the modularity value should increase because of the split. Therefore,

$$
\begin{aligned}
&Q'_c < \sum_{i=1}^{k} Q_{X_i} \\
&\Leftrightarrow \frac{\sum_{i=1}^{k} m_{X_i} + \sum_{i<j} e_{ij} + 1}{M+1} - \frac{(\sum_{i=1}^{k} D_{X_i+2})^2}{4(M+1)^2} \\
&< \frac{\sum_{i=1}^{k} m_{X_i} + 1}{M+1} - \frac{(D_{X_1}+2)^2}{4(M+1)^2} - \sum_{i=2}^{k} \frac{D_{X_i}^2}{4(M+1)^2} \\
&\Leftrightarrow \sum_{i<i} e_{ij} < \frac{\sum_{i=1}^{k} D_{X_i} - 2D_{X_1} + \sum_{i<j} D_{X_i} D_{X_j}}{2(M+1)}
\end{aligned}
$$

Since $\sum_{i=1}^{k} D_{X_i} - 2D_{X_1} < 2M$, this implies that

$$
\begin{aligned}
\frac{\sum_{i<j} D_{X_i} D_{X_j}}{2M} < \sum_{i<j} e_{ij} &< \frac{\sum_{i=1}^{k} D_{X_i} - 2D_{X_1} + \sum_{i\neq j} D_{X_i} D_{X_j}}{2(M+1)} \\
&< \frac{\sum_{i<j} D_{X_i} D_{X_j}}{2M} + 1
\end{aligned}
$$

Therefore, the proposition holds. □

PROPOSITION 3. *If a new edge (u, v) connecting two communities $C(u)$ and $C(v)$ is introduced, $C(u)$ (or $C(v)$) is the best candidate for v (or u) if it should ever change its membership.*

PROOF. The method is inspired by [39] that a vertex u is influenced by two factors: $F_{in}^{c}(u)$ the force that keeps u stay in its own community c, and $F_{out}^{c}(u)$, the force that a community S imposes to u in order to bring u to S as follows:

$$F_{in}^{c}(u) = e_{c}^{u} - \frac{d_u(D_c - d_u)}{2M}$$

and

$$F_{out}^{S}(u) = max_{S \in NC(u)}\{e_{S}^{u} - \frac{d_u D_{outS}}{2M}\}$$

where $NC(u)$ is the set of neighboring communities of u, and D_{outS} is the total degree of vertices outside S.

Now we will show that the presence of new edge (u, v) will strengthen $F_{out}^{C(v)}(u)$ and weaken $F_{out}^{S}(u)$. In other words, we will show that $F_{out}^{C(v)}(u)$ increases while $F_{out}^{S}(u)$ decreases for all $S \in C \wedge S \notin \{C(u), C(v)\}$.

$$
\begin{aligned}
&F_{out}^{C(v)}(u)|_{new} - F_{out}^{C(v)}(u)|_{old} \\
&= (e_{u}^{C(v)} + 1 - \frac{(d_u+1)(d_{outC(v)}+1)}{2(M+1)} - (e_{u}^{C(v)} - \frac{d_u d_{outC(v)}}{2M}) \\
&\geq \frac{2M + d_u d_{outC(v)}}{2(M+1)} - \frac{d_u d_{outC(v)} + d_{outC(v)} + d_u + 1}{2(M+1)} \\
&> 0
\end{aligned}
$$

Therefore $F_{out}^{C(v)}(u)$ is strengthened when a new edge (u, v) is introduced. Further, for any community $S \in C \wedge S \notin \{C(u), C(v)\}$

$$
\begin{aligned}
&F_{out}^{S}(u)|_{new} - F_{out}^{S}(u)|_{old} \\
&= (e_{u}^{S} - \frac{(d_u+1)d_{outS}}{2(M+1)}) - (e_{u}^{S} - \frac{d_u d_{outS}}{2M}) \\
&= d_{outS}(\frac{d_u}{2M} - \frac{d_u+1}{2(M+1)}) < 0
\end{aligned}
$$

This implies that $F_{out}^{S}(u)$ is weakened when (u, v) is added. Therefore, the proposition holds. □

PROPOSITION 4. *If a new edge (u, v) is added into the graph, then joining u to v's community $C(v)$ will increase the modularity value if $\Delta Q(u, C(u), C(v)) \equiv 4(M+1)(e_{C(v)}^{u} + 1 - e_{C(u)}^{u}) + e_{C(v)}^{u}(2D_{C(v)} - 2d_u - e_{C(u)}^{u}) - 2(d_u+1)(d_u + 1 + d_{C(v)} - d_{C(u)}) > 0$.*

PROOF. Vertex u will leave its current community $C(u)$ and join v's community $C(v)$ if

$$
\begin{aligned}
&Q_{C(v)+u} + Q_{C(u)-u} > Q_{c(u)} + Q_{C(v)} \\
&\Leftrightarrow \frac{m_{C(v)} + e_{C(v)} + 1}{M+1} - \frac{(d_{C(v)} + d_u + 2)^2}{4(M+1)^2} + \\
&\frac{m_{C(u)} - e_{C(u)}}{M+1} - \frac{(d_{C(u)} - d_u - e_{C(u)})}{4(M+1)^2} \\
&> \frac{m_{C(v)}}{M+1} - \frac{(d_{C(v)} + d_u + 2)^2}{4(M+1)^2} + \frac{m_{C(u)}}{M+1} - \frac{(d_{C(u)}+1)^2}{4(M+1)^2} \\
&\Leftrightarrow 4(M+1)(e_{C(v)} + 1 - e_{C(u)}) + e_{C(u)}(2d_{C(v)} - 2d_{C(u)} - e_{C(u)}) \\
&- 2(d_{C(u)}+1)(d_{C(u)} + 1 + d_{C(v)} - d_{C(u)}) > 0
\end{aligned}
$$

□

Corollary 1. *If the condition in Proposition 4 is not satisfied, then neither u nor its neighbors should be assigned to $C(v)$.*

Function 1: *BothinSample*$(u, v, e_t, V_s, E_s, C_s)$

```
1  if C_s(u) == C_s(v) then
2      E_s = E_s ∪ e_t
3  else
4      if ΔQ(u, C_s(u), C_s(v)) < 0 ∧ ΔQ(v, C_s(v), C_s(u)) < 0 then
5          return V_s, E_s, C_s
6      else
7          w ← argmax{ΔQ(u, C_s(u), C_s(v)), ΔQ(v, C_s(v), C_s(u))}
8          Move w to a new community and update C_S
9          for t ∈ N(w) do
10             Let t decide its own community
11             Update C_s
12 return V_s, E_s, C_s
```

(ii) **u is in sample and v is new:** When a new node v connecting node $u \in G_s$ appears, *OneinSampleOneNew*() (see Function 2) is called from line 17 of Algorithm 1 (edge $\{b, p\}$ in Figure 1). In this case, we do not add $\{u, v\}$ into the sample immediately. Instead, we first insert v into $\mathcal{H}$ if $\mathcal{H}$ is not full. If $\mathcal{H}$ is full, we pick one node x from $\mathcal{H}$ preferentially based on $\mathcal{H}_c[x]$ with an additional constraint that $\mathcal{P}(x)$, the parent of x should be in G_s[4] (for example, in Figure 1 if the buffer is full although node l has highest count we do not pick l from the buffer because its parent node i is not in G_s; instead we pick m which satisfies both the constraints). x is then added to G_s through the edge $\{\mathcal{P}(x), x\}$ (line 10 in Function 2) and is assigned the community of $\mathcal{P}(x)$ (line 11 in Function 2). We also check if including x in G_s violates the sample size constraint through the function *CheckResizeSample*() (Function 7). If G_s is already full (i.e., $V_s = n$), one existing node which has lowest degree is removed from G_s and the community structure is adjusted accordingly using *CommunityAfterNodeRemoval*() (both Functions 7 and 8 are discussed later).

(iii) **u is in sample and v is in buffer:** In this case (edge $\{d, i\}$ in Figure 1), *OneinSampleOneinBuffer*() (see Function 3) is called from line 19 of Algorithm 1. We first check

[4]The additional constraint allows to avoid a disconnected component being created in the sample.

Function 2: $OneinSampleOneNew(u, v, e_t, V_s, E_s, \mathcal{H}, C_s)$

```
1  if H is not full then
2      Insert v to H
3  else
4      Choose a node x with P(x) ∈ V_s from H preferentially
       based on H_c
5      Remove x from H
6      H_c[x] = 0
7      H_p[x] = φ
8      V_s, E_s, C_s = CheckResizeSample(V_s, C_s, n, 1)
9      V_s = V_s ∪ x
10     E_s = E_s ∪ {P(x), x}
11     C_s(x) = C_s(P(x))
12     Update C_s
13     Insert v to H
14 H_c[v] = 1
15 H_p[v] = u
16 return V_s, E_s, C_s, H
```

the sample size constraint (line 4 in Function 3) and accordingly add v (and edge $\{u, v\}$) into G_s. v is further assigned to $C(u)$.

Function 3: $OneinSampleOneinBuffer(u, v, e_t, V_s, E_s, \mathcal{H}, C_s)$

```
1  Remove v from H
2  H_c[v] = 0
3  H_p[v] = φ
4  V_s, E_s, C_s = CheckResizeSample(V_s, C_s, n, 1)
5  V_s = V_s ∪ v
6  E_s = E_s ∪ {v, P(v)}
7  C_s(v) = C_s(u)
8  Update C_s
9  return V_s, E_s, C_s, H
```

(iv) u is in buffer and v is new: This case (edge $\{m, p\}$ in Figure 1, see Function 4) is similar to Function 2. We first increment the counter corresponding to u in the buffer and add v into the buffer in the same way as mentioned in Function 2.

Function 4: $OneinBufferOneNew(u, v, e_t, V_s, E_s, \mathcal{H}, C_s)$

```
1  H_c[u] = H_c[u] + 1
2  V_s, E_s, C_s, H = OneinSampleOneNew(u, v, e_t, V_s, E_s, H, C_s)
3  return V_s, E_s, C_s, H
```

(v) Both u and v are new: When both u and v are new (edge $\{p, q\}$ in Figure 1), $BothNew()$ (see Function 5) is called from line 23 of Algorithm 1. We initially check whether buffer $\mathcal{H}$ is full or not. In case it is not full, we insert u in $\mathcal{H}$ and then call Function 2. Otherwise, we preferentially remove x and y from $\mathcal{H}$ based on the counts in $\mathcal{H}_c$ with the additional constraint that both $\mathcal{P}(x)$ and $\mathcal{P}(y)$ are in G_s, and add nodes x, y and edges $\{\mathcal{P}(x),x\}$, $\{\mathcal{P}(y),y\}$ into G_s. x and y are also assigned to the community of the their respective parents. Finally, u and v are inserted into $\mathcal{H}$. If during the insertion into G_s the sample size is violated, the required number of nodes along with their adjacent edges are deleted from G_s using $CheckResizeSample()$.

(vi) Both u and v are in buffer: In this case (edge $\{k, n\}$ in Figure 1), $BothinBuffer()$ (see Function 6) is called from line 24 of Algorithm 1 whereby, only the buffer $\mathcal{H}$ is modified by increasing $\mathcal{H}_c$ entries of u and v by 1.

• Adjust communities after removing a node: When G_s is full, we remove m nodes (and their adjacent edges) from the sample using $CheckResizeSample()$ (Function 7). Since each such node u is already a part of its community $C(u)$, its deletion might keep the previous community structure unchanged, or break the community into smaller parts, or merge several communities together. The community structure C_s is adjusted using $CommunityAfterNodeRemoval()$ (Function 8) incrementally. Let us consider two extreme cases – a node with degree 1 is removed, and a node with highest degree is removed. Removal of a single degree node will keep the community unchanged. However, removal of a highest degree node can render the community disconnected or broken into smaller parts which might further merge to the other existing communities. Here we utilize the clique percolation method [29] to handle this situation efficiently. In particular, when a vertex v is removed from a community C, we place a 3-clique to one of its neighbors and let the clique percolate until no vertices in C are discovered. Nodes discovered in each such clique percolation will form a community. We repeat this clique percolation from each of v's neighbors until each member in C is assigned to a community. For example, in Figure 2 when node g is removed, we place a 3-clique on its neighbor a. Once the 3-clique starts percolating, it accumulates all nodes except f. Therefore, two new communities $\{a, b, c, d, e\}$ and $\{f\}$ emerge due to the deletion of g. In this way, we let the remaining communities of C choose their best communities to merge in.

Function 5: $BothNew(u, v, e_t, V_s, E_s, \mathcal{H}, C_s)$

```
1  if H is not full then
2      Insert u to H
3      H_p[u] = v
4      H_c[u] = 1
5      V_s, E_s, C_s, H =
       OneinSampleOneNew(u, v, e_t, V_s, E_s, H, C_s)
6  else
7      Choose nodes x, y with P(x), P(y) ∈ V_s from H
       preferentially based on H_c
8      Remove x, y from H
9      H_c[x] = 0, H_c[y] = 0
10     H_p[x] = φ, H_p[y] = φ
11     V_s, E_s, C_s = CheckResizeSample(V_s, C_s, n, 2)
12     V_s = V_s ∪ x ∪ y
13     E_s = E_s ∪ {x, P(x)} ∪ {y, P(y)}
14     C_s(x) = C_s(P(x))
15     C_s(y) = C_s(P(y))
16     Update C_s
17     Insert u, v to H
18     H_p[u] = v, H_p[v] = u
19     H_c[u] = 1, H_c[v] = 1
20 return V_s, E_s, C_s, H
```

Function 6: $BothinBuffer(u, v, \mathcal{H})$

```
1  H_c[u] = H_c[u] + 1
2  H_c[v] = H_c[v] + 1
3  return H
```

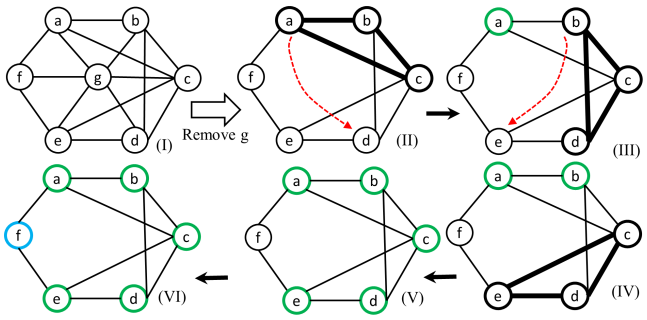

Figure 2: Illustrative example of 3-clique percolation. Once node g is removed, a 3-clique is placed on node a. The clique percolates and accumulates all the nodes except node f which forms a singleton community along with $\{a, b, c, d, e\}$.

Function 7: *CheckResizeSample*(V_s, C_s, n, m)

```
if V_s == n then
    Remove m nodes say, u_1, u_2, ..., u_m (and all their adjacent
    edges) from G_s having lowest degree
    for u ∈ {u_1, u_2, ..., u_m} do
        C_s ← CommunityAfterNodeRemoval(u, C_s)
return V_s, E_s, C_s
```

Function 8: *CommunityAfterNodeRemoval*(u, C_s)

```
Assume node u and its adjacent edges are removed from G_s
i = 1
while N(u) ≠ φ do
    b_i=Nodes found by a 3-clique percolation on v ∈ N(u)
    if b_i == φ then
        b_i = {v}
    C_s = C_s ∪ b_i
    N(u) = N(u) \ b_i
    i = i + 1
Update C_i
return C_s
```

5. EXPERIMENTAL SETUP

In this section, we describe the baseline sampling algorithms and the datasets used in our experiments.

5.1 Sampling algorithms

We compare our method with five existing sampling methods: (i) Streaming Node (SN) [4], (ii) Streaming Edge (SE) [4], (iii) Streaming BFS (SBFS) [4], (iv) PIES [3], and (v) Green algorithm (GA) [35]. The first four algorithms are exclusively designed for streaming graphs while the last one is designed for static graphs. Note that unlike ours, none of the existing algorithms explicitly produce a community structure as a bi-product of the sampling[5], and thus one needs to execute community detection algorithm separately on the sample to obtain the community structure. Therefore to evaluate the competing algorithms with respect to how the underlying community structure in the sample resembles with that of the original graph, for SN, SE, SBFS and PIES we run Louvain algorithm [5] on each individual sample and detect the communities. In case of GA, we consider the aggregated graph and run GA to obtain the sample, and further run Louvain on the sample to detect the community structure. Note that although by considering the aggregated graph GA acquires far more information of the entire graph structure, we use it as a strict baseline in this study to show that the community structure obtained from `ComPAS` is quite competitive to that obtained by running Louvain on GA's sample graph.

5.2 Datasets

We perform our experiments on five graphs mentioned below. While first two are streaming graphs, last three graphs are static graphs.

(i) **Facebook**[6]: This is an undirected graph with nodes representing users, and an edge exists between two users if they are friends. Each edge is time stamped by the occurrence of the friendship. The graph consists of $63,731$ nodes and $817,035$ edges.

(ii) **arxiv hep-th**[7]: This dataset represents the collaboration graph of the authors in arXiv's High Energy Physics papers. Each node represents an author and an edge exists between two authors if they have co-authored a paper. The edges are time stamped by publication date of the co-authored paper. The graph consists of 22,908 nodes and 2,673,133 edges.

(iii) **Youtube**[8]: This dataset represents the Youtube social network with nodes representing users and edges representing friendship. The graph consists of 1,134,890 nodes and 2,987,624 edges.

(iv) **Dblp**[9]: This dataset consists of authors indexed in DBLP. The graph is same as arxiv hep-th. There are 317,080 nodes and 1,049,866 edges in the graph.

(v) **LFR**: This is a synthetic graph [22] with underlying community structure implanted into it. We construct the graph with 25,000 nodes, 254,402 edges and 18,34 communities.

Since last three graphs are static, we consider that each edge arrives in a pre-decided order (which is set randomly) i.e., each edge has a (discrete) time of arrival. We later show that edge ordering does not influence the inferences drawn from the results (Section 6.3).

Moreover, since first four graphs do not have any underlying ground-truth community structure, we run Louvain algorithm [5] on the aggregated graph and obtain the disjoint community structure. This community structure is the best possible output that we can expect from our incremental modularity maximization method, and therefore serves as the ground-truth.

6. EVALUATION

We design a two-fold experimental setup. First, we show how competing sampling algorithms detect the original community structure, and second, we measure how good individual samples are w.r.t. the structural properties of the original graph. Although the primary focus of `ComPAS` is to quality better in the first evaluation, we also show that `ComPAS` is quite competitive to retain the graph structure in the second evaluation.

6.1 Community-centric evaluation

In this section, we start by explaining the metrics used to evaluate the goodness of the community structure, followed by a detailed comparison of the sampling algorithms.

6.1.1 Evaluation criteria

To measure how sampling algorithms capture the underlying community structure, we evaluate them in two ways. First we measure the quality of the obtained community structure based on the **topological measures** defined by Yang et. al. [38]. In particular, we look into four classes of quality scores - (i) *based on internal connectivity*: internal density (ID), edge inside (EI), average degree (AD), fraction over mean degree (FOMD), triangle participation ratio (TPR); (ii) *based on external connectivity*: expansion (EX), cut ratio (CR); (iii) *combination of internal and external connectivity*: conductance (CON), normalized cut (NC), maximum out-degree fraction (MODF), average out-degree fraction (AODF), flake out-degree fraction (FODF); and (iv)

[5]Although GA claims that its sample graph preserves the underlying community structure, it does not explicitly produce the community structure.

[6]konect.uni-koblenz.de/networks/facebook-wosn-links

[7]konect.uni-koblenz.de/networks/ca-cit-HepTh

[8]snap.stanford.edu/data/com-Youtube.html

[9]snap.stanford.edu/data/com-DBLP.html

Table 2: Summary of the D-statistics (the lower, the better) values of the topological measures for all the datasets. For Youtube we present all the results, while for the rest we provide the average D-statistics and standard deviation (SD). ComPAS truns out to the second best algorithm after GA (the most informed static graph sampling algorithm for which the sample is obtained from the aggregated graph and Louvain is run on the sample, thus serving as the strict baseline). Top two values for each average result is highlighted.

Algorithm	Youtube														Facebook	Com-dblp	LFR	hep-th
	ID	EI	AD	FOMD	TPR	EX	CR	CON	NC	AODF	MODF	FODF	MOD	Avg,SD	Avg,SD	Avg,SD	Avg,SD	Avg,SD
ComPAS	0.063	0.051	0.078	0.057	0.227	0.082	0.054	0.091	0.260	0.073	0.201	0.121	0.052	**0.10,0.07**	**0.17,0.09**	**0.16,0.10**	**0.18,0.06**	**0.10,0.03**
SN	0.164	0.171	0.471	0.061	0.542	0.581	0.112	0.265	0.064	0.157	0.182	0.092	0.216	0.23,0.17	0.33,0.17	0.29,0.20	0.27,0.07	0.26,0.04
SE	0.257	0.244	0.241	0.501	0.281	0.098	0.287	0.087	0.151	0.097	0.246	0.093	0.198	0.21,0.11	0.27,0.11	0.25,0.14	0.32,0.08	0.29,0.06
SBFS	0.126	0.131	0.172	0.106	0.454	0.145	0.056	0.165	0.045	0.257	0.108	0.076	0.181	0.15,0.10	0.26,0.09	0.24,0.10	0.25,0.09	0.26,0.04
PIES	0.234	0.241	0.252	0.190	0.409	0.042	0.051	0.049	0.061	0.157	0.042	0.053	0.121	0.14,0.10	0.29,0.06	0.24,0.07	0.26.0.05	0.21,0.05
GA	0.156	0.055	0.065	0.053	0.267	0.066	0.076	0.053	0.085	0.150	0.075	0.069	0.102	**0.09,0.06**	**0.12,0.04**	**0.12,0.06**	**0.14,0.06**	**0.08,0.04**

based on graph model: modularity (MOD)[10]. Note for every individual community we obtain a score, and therefore a distribution of scores (i.e., distribution of ID, distribution of EI etc.) is obtained for all the communities of a graph. We measure how similar (in terms of Kolgomorov-Smirnov D-statistics[11]) these distributions are with those of the ground-truth communities. *The less the value of D-statistics, the better the match between two distributions.*

As a second level of evaluation, we use the **community validation metrics** – Purity [26], Normalized Mutual Information (NMI) [7] and Adjusted Rand Index (ARI) [19] to measure the similarity between the ground-truth and the obtained community structures. *The more the value of these metrics, the more the similarity.*

6.1.2 Parameter estimation

As reported in Section 4, our algorithm consists of two parameters (i) α (initial fraction of nodes inserted), (ii) n_d (length of the buffer). For α, we observe that D-statistics is initially high and reduces as we increase α (Figures 3(a)). This is because for low α values the community structure obtained initially by running Louvain algorithm (Step 12 in Algorithm 1) is coarse. For large values of α even though initial community structure obtained is good, it is not allowed to evolve much. Similarly in Figure 3(b), given a small buffer size several nodes mostly arriving once would be added to the sample leading to formation of pendant vertices. As we increase the buffer size ComPAS performs better till a certain point, after which the improvement is negligible. Since we are interested in using minimum space we fix n_d at $0.001 \cdot n$. Thus, for the rest of the experiments we set α to 0.5 and n_d to $0.001 \cdot n$ unless otherwise specified. Further we set n to $0.4|V|$ as default (see Section 6.3 for different values of n).

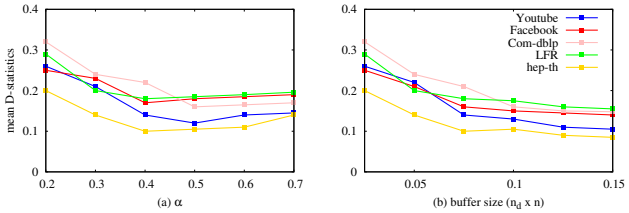

Figure 3: Average D-statistics value across all the topological measures.

6.1.3 Comparison of sampling algorithms

We start by measuring the similarity between the obtained and the ground-truth community structures using topological measures. In Table 2 we summarize the D-statistics values of all the scoring functions for Youtube dataset; for the other graphs we only present the average value (and standard deviation) across the D-statistics among different topological measures (see [1] for detailed results). Clearly ComPAS outperforms all the streaming algorithms across different datasets. GA performs better than ComPAS since it has in its consideration the whole graph to obtain the sample. However, we stress that even with *minimal community information to start with* and *no subsequent community detection in the later steps*, we are able to reach very close to GA as well as to the ground-truth.

Table 3: NMI between the ground-truth and community structure obtained from individual sampling algorithms for all datasets.

Dataset	ComPAS	SN	SE	SBFS	PIES	GA
Facebook	0.52	0.34	0.28	0.41	0.48	0.61
hep-th	0.51	0.32	0.21	0.36	0.39	0.68
Youtube	0.72	0.49	0.33	0.58	0.51	0.77
Dblp	0.65	0.28	0.21	0.57	0.39	0.69
LFR	0.69	0.29	0.32	0.38	0.31	0.72
Average	**0.61**	0.34	0.27	0.46	0.41	**0.69**

As a second level of evaluation, we further calculate three validation metrics – purity, NMI and ARI between the ground-truth and the obtained community structure for all the algorithms (see Table 3 for NMI, details in [1]). Once again we observe that ComPAS is the second ranked algorithm after GA with an average (over all datasets) purity, NMI and ARI of 0.74, 0.61 and 0.53 respectively.

6.2 Graph-centric evaluation

Although our primary objective is to obtain a community-preserved graph sample, we further evaluate our sample in terms of three graph properties mentioned in [4]: (i) degree distribution (Degree), (ii) clustering coefficient distribution (CC), (iii) top 100 eigen value distribution (EV). In Table 4 we report the D-statistics values between the distributions of the original graph and those obtained from the sample (see more results in [1]). For all the datasets, we observe that ComPAS stands within top three ranks in terms of low D-statistics, in many cases, also beating the strict baseline GA. This essentially indicates that ComPAS, apart from preserving the community structure, also preserves general graphs properties in the sample.

6.3 Effect of edge ordering and sample size

As mentioned in Section 5.2, we artificially imposed an edge ordering for two static graphs – Youtube and LFR. In this section, we show that most of our inferences are valid irrespective of any edge ordering. To do so, we randomly pick one pair of edges and swap their arrival time. We repeat it for 20% of edges present in each static graph. The entire experiment is repeated 20 times and the average value is

[10]See [38] for the detailed definitions of all these metrics.

[11]It is defined as $D = max_x\{|f(x) - f^{'}(x)|\}$ where x is over the range of the random variable, and f and $f^{'}$ are the two empirical cumulative distribution functions of the data.

Table 4: Summary of D-statistics for different graph properties. For Youtube we present all the results, while for the rest we provide only average D-statistics (top three results in each average case are highlighted).

Algorithm	Youtube				Facebook	Com-dblp	LFR	hep-th
	Degree	CC	EV	Average	Average	Average	Average	Aveage
ComPAS	0.083	0.105	0.42	**0.20**	**0.17**	**0.16**	**0.28**	**0.18**
SN	0.076	0.108	0.54	0.24	0.36	0.24	0.34	0.23
SE	0.114	0.195	0.39	0.23	0.48	0.28	0.39	0.27
SBFS	0.105	0.172	0.32	**0.19**	**0.28**	**0.13**	**0.26**	**0.19**
PIES	0.046	0.129	0.38	**0.18**	**0.16**	0.21	**0.28**	**0.16**
GA	0.218	0.063	0.46	0.24	0.35	**0.12**	0.32	0.20

reported. Figure 4(a) shows that for Youtube graph edge ordering does not affect much the final sample obtained (the pattern is same for LFR graph, see [1]).

Lastly, we present the effect of sample size (n) on the obtained community structure. We plot average D-statistics values across all the topological measures for all the algorithms on Youtube dataset (see others in [1]) as a function of n (Figure 4(b)). As expected, with the increase of n we obtain better results. Interestingly, for ComPAS and GA, the pattern remains consistent compared to others.

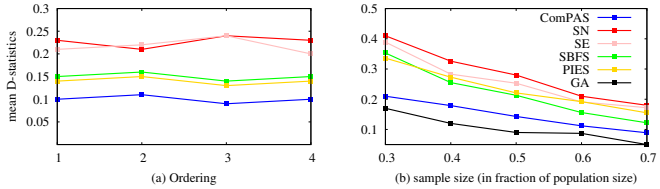

Figure 4: Average D-statistics across all the topological measures for (a) different edge ordering and (b) sample size (n) of Youtube graph.

7. APPLICATIONS OF COMPAS

Here we present two additional applications of ComPAS – ranking community detection algorithms, and selecting appropriate training set for (restricted) online learning.

7.1 Ranking community detection algorithms

As pointed out in [24], there is always a trade-off between quality and running time for discovering the community structure from a graph. While some algorithms discover excellent community structure with larger running time, others compromise quality by achieving lesser running time. Further, it is not possible to identify beforehand which community detection algorithm is more useful given a large graph. In this case, one is forced to run several algorithms on the large graph and converge on the best performing one. This may lead to a large computation cost and is especially detrimental in resource-scarce situations (absence of parallel infrastructures, distributed systems etc.).

In case of community detection on large streaming graphs, we posit that one may first execute ComPAS and obtain a community-preserved sample. Then different community detection algorithms can be tested on the sample (which is much smaller in size compared to the original graph) and the best performing algorithm can be used for the original large graph. To this end, we execute five community detection (CD) algorithms (CNM [6], Fast-Greedy [28], Walktrap [30], Infomod [33], Infomop [34]) separately on each sample obtained from the sampling algorithms. Then we rank the CD algorithms based on their performance in terms of modularity [27]. We further rank the CD algorithms in the same way considering the *entire graph*. We report the Kendall's τ rank correlation in Table 5(a) between these two ranks for individual sampling algorithms across all the datasets. We find GA and ComPAS performing the best. Therefore, ComPAS can be used for selecting a CD algorithm quickly before running on a large streaming graph.

Table 5: Rank correlation of community detection algorithms based on the performance on the sample (generated from individual sampling methods) and the original graph. (b) Performance of SVM using the training set obtained from sampling methods.

(a)

Algo	Facebook	hep-th	Youtube	DBLP	LFR
ComPAS	**0.8**	**0.8**	**0.8**	**1.0**	**1.0**
SN	0.4	0.6	0.4	0.6	0.6
SE	0.4	0.4	0.4	0.6	0.4
SBFS	0.6	0.7	0.6	0.8	0.6
PIES	0.6	0.6	0.8	0.7	0.8
GA	**1.0**	**0.8**	**0.8**	**1.0**	**1.0**

(b)

AUC	F-Score
0.48	**0.61**
0.31	0.35
0.25	0.28
0.28	0.31
0.36	0.43
0.53	**0.64**

7.2 Selecting training set for online learning

In online learning, sometimes memory is limited and it is required to train the model on limited number of instances. In such situations it is important to choose a training set in such a way that it consists of members from most or all parts of the original dataset.

We hypothesize that more diverse the chosen set, better would be the performance. ComPAS is useful in such cases since it tries to sample from several communities, hence improving the diversity of the training set. To this end, we consider Wiki-Rfa[12] [37], a streaming signed graph in which nodes represent Wikipedia members and edges (with timestamp) represent votes. Each vote is typically accompanied by a short comment. The task is to predict the vote (+1, 0, -1) of an incoming edge based on the textual features – (i) word count, (ii) sentiment value, and (iii) LIWC features of the statement corresponding to the edge. We allow training instances to be included till a certain time period t (first 75% of the edges are allowed to enter) and run the sampling algorithms in parallel. However not all instances can be considered for training due to the memory constraint. We assume n, the sample size as the allowed training size and obtain sampled training set from individual sampling algorithms. We train SVM with linear kernel (see [1] for other classifiers) on each sampled training set, and predict the labels (votes) of those instances coming after t. Table 5(b) shows that GA and ComPAS perform the best in terms of AUC and F-Score. This once again emphasizes the fact that ComPAS selects most representative training instances for (restricted) online learning.

8. CONCLUSION

In this paper we proposed ComPAS, a novel sampling algorithm for streaming graphs which is able to retain the community structure of the original graph. Through rigorous experimentation on four real-world and one synthetic graphs we showed that our algorithm performs better than five state-of-the-art graph sampling algorithms. Further we showed the effectiveness of ComPAS through two real-world applications. As an immediate future work we plan to analytically estimate the relation between sample size and the accuracy of our algorithm. It would also be interesting to investigate in detail the effectiveness of ComPAS in other real applications.

[12]https://snap.stanford.edu/data/wiki-RfA.html

9. REFERENCES

[1] Supplementary Materials. http://tinyurl.com/SI-ComPAS. [Online; accessed 24-Oct-2016].

[2] C. C. Aggarwal, Y. Zhao, and S. Y. Philip. Outlier detection in graph streams. In *ICDE*, pages 399–409. IEEE, 2011.

[3] N. K. Ahmed, J. Neville, and R. Kompella. Reconsidering the foundations of network sampling. In *Proceedings of the 2nd Workshop on Information in Networks*, 2010.

[4] N. K. Ahmed, J. Neville, and R. Kompella. Network sampling: From static to streaming graphs. *ACM TKDD*, 8(2):7, 2014.

[5] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008, 2008.

[6] A. Clauset, M. E. Newman, and C. Moore. Finding community structure in very large networks. *Physical review E*, 70(6):066111, 2004.

[7] L. Danon, A. Diaz-Guilera, J. Duch, and A. Arenas. Comparing community structure identification. *Journal of Statistical Mechanics: Theory and Experiment*, 2005(09):P09008, 2005.

[8] A. Doucet, S. Godsill, and C. Andrieu. On sequential monte carlo sampling methods for bayesian filtering. *Statistics and computing*, 10(3):197–208, 2000.

[9] S. Fortunato. Community detection in graphs. *Physics reports*, 486(3):75–174, 2010.

[10] O. Frank. Survey sampling in graphs. *Journal of Statistical Planning and Inference*, 1(3):235–264, 1977.

[11] O. Frank. Sampling and inference in a population graph. *International Statistical Review*, pages 33–41, 1980.

[12] K. J. Gile and M. S. Handcock. Respondent-driven sampling: An assessment of current methodology. *Sociological methodology*, 40(1):285–327, 2010.

[13] M. Girvan and M. E. Newman. Community structure in social and biological networks. *PNAS*, 99(12):7821–7826, 2002.

[14] M. Gjoka, M. Kurant, C. T. Butts, and A. Markopoulou. Walking in facebook: a case study of unbiased sampling of osns. In *Infocom*, pages 1–9. IEEE, 2010.

[15] L. A. Goodman. Snowball sampling. *The annals of mathematical statistics*, pages 148–170, 1961.

[16] W. K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

[17] D. D. Heckathorn. Respondent-driven sampling: a new approach to the study of hidden populations. *Social problems*, 44(2):174–199, 1997.

[18] M. R. Henzinger, P. Raghavan, and S. Rajagopalan. Computing on data streams. Technical report, Technical Note 1998-011, Digital Systems Research Center, Palo Alto, CA, 1998.

[19] L. Hubert and P. Arabie. Comparing partitions. *Journal of classification*, 2(1):193–218, 1985.

[20] E. D. Kolaczyk. *Statistical Analysis of Network Data: Methods and Models*. Springer, 1st edition, 2009.

[21] E. Kolascyk. Statistical analysis of network data. *SAMSI program on Complex networks. Boston university*, 2013.

[22] A. Lancichinetti, S. Fortunato, and F. Radicchi. Benchmark graphs for testing community detection algorithms. *Physical review E*, 78(4):046110, 2008.

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

[24] A. S. Maiya and T. Y. Berger-Wolf. Sampling community structure. In *WWW*, pages 701–710. ACM, 2010.

[25] A. S. Maiya and T. Y. Berger-Wolf. Benefits of bias: Towards better characterization of network sampling. In *ACM SIGKDD*, pages 105–113. ACM, 2011.

[26] C. D. Manning, P. Raghavan, and H. Schütze. Introduction to information retrieval, 2008.

[27] M. E. Newman. Analysis of weighted networks. *Physical review E*, 70(5):056131, 2004.

[28] M. E. Newman. Fast algorithm for detecting community structure in networks. *Physical review E*, 69(6):066133, 2004.

[29] G. Palla, I. Derenyi, I. Farkas, and T. Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 435(7043):814–818, 2005.

[30] P. Pons and M. Latapy. Computing communities in large networks using random walks. In *International Symposium on Computer and Information Sciences*, pages 284–293. Springer, 2005.

[31] A. H. Rasti, M. Torkjazi, R. Rejaie, N. Duffield, W. Willinger, and D. Stutzbach. Respondent-driven sampling for characterizing unstructured overlays. In *INFOCOM*, pages 2701–2705. IEEE, 2009.

[32] B. Ribeiro and D. Towsley. Estimating and sampling graphs with multidimensional random walks. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 390–403. ACM, 2010.

[33] M. Rosvall and C. T. Bergstrom. An information-theoretic framework for resolving community structure in complex networks. *PNAS*, 104(18):7327–7331, 2007.

[34] M. Rosvall and C. T. Bergstrom. Maps of random walks on complex networks reveal community structure. *PNAS*, 105(4):1118–1123, 2008.

[35] C. Tong, Y. Lian, J. Niu, Z. Xie, and Y. Zhang. A novel green algorithm for sampling complex networks. *Journal of Network and Computer Applications*, 59:55–62, 2016.

[36] S. Wasserman and K. Faust. *Social network analysis: Methods and applications*, volume 8. Cambridge university press, 1994.

[37] R. West, H. S. Paskov, J. Leskovec, and C. Potts. Exploiting social network structure for person-to-person sentiment analysis. *arXiv preprint arXiv:1409.2450*, 2014.

[38] J. Yang and J. Leskovec. Defining and evaluating network communities based on ground-truth. *KIAS*, 42(1):181–213, 2015.

[39] Z. Ye, S. Hu, and J. Yu. Adaptive clustering algorithm for community detection in complex networks. *PRE*, 78:046115, Oct 2008.