Information Sciences 482 (2019) 279–300

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Cluster-preserving sampling from fully-dynamic streaming graphs

Jianpeng Zhang[a,b,*], Kaijie Zhu[a,b], Yulong Pei[a], George Fletcher[a], Mykola Pechenizkiy[a]

[a] Eindhoven University of Technology, 5600 MB, Eindhoven, the Netherlands
[b] National Digital Switching System Engineering & Technological R&D Center, Zhengzhou 450002, China

ARTICLE INFO

Article history:
Received 22 January 2018
Revised 3 January 2019
Accepted 4 January 2019
Available online 5 January 2019

Keywords:
Graph sampling
Fully-dynamic streaming graph
Clustering structure
Reservoir sampling
Isolated nodes

ABSTRACT

Current sampling techniques on graphs (i.e., network-structured data) mainly study static graphs and suppose that the whole graph is available at all times. However, a surge of graphs are becoming too large-scale and/or fully-dynamic (i.e., nodes and edges are added or deleted arbitrarily) to be handled with small memory footprint. Moreover, the intrinsic property (i.e., clustering structure) has been ignored and is not preserved well when the sampling performs. To solve these issues, we propose a Cluster-preserving Partially Induced Edge Sampling (*CPIES*) algorithm that dynamically keep up-to-date samples in an online fashion and preserve the inherent clustering structure in streaming graphs. The experiments on various synthetic and real-world graphs demonstrated that the proposed *CPIES* algorithm is capable of preserving the inherent clustering structure while sampling from the fully-dynamic streaming graphs, and performs better than other competing algorithms in cluster-related properties.

1. Introduction

Graph, as a generic data structure, can be a good representation of complex relationships among entities of networks. For instance, in the World Wide Web, web pages can be represented by nodes and the hyperlinks between these pages are denoted by edges. With the advent of the era of big data, a large number of graphs are becoming large-scale and ever-increasing in nature. For example, in Twitter network average monthly active users have increased to 328 million in the 1st quarter of 2017, adding close to 9 million users than the preceding quarter, thus some of the properties (e.g., clustering structure) of the graph may have correspondingly changed over time. Hence, graph analysis on such data as a whole is becoming more and more impractical and time-consuming. Generally, there are mainly two reasons that make it difficult to study these graphs in their entirety. Firstly, the massive sizes of many graphs make them computationally expensive to analyze the whole graphs. Moreover, designing more efficient algorithms and/or leveraging computing power are not always easily available [2,5,19]. Secondly, due to the limitations in data collection mechanism, contemporary graphs that are considered as complete graphs are not completely accessible and partially visible to the users [7,11]. One approach to overcome these features of contemporary graph-structured data collections is *sampling*, i.e., to sample a representative subgraph and exploit its characteristics. It not only provides an efficient solution in big-graph analytics when the population

* Corresponding author.
E-mail address: j.zhang.4@tue.nl (J. Zhang).

https://doi.org/10.1016/j.ins.2019.01.011
0020-0255/

is infeasible on account of the inaccessibility and massive-size, but also is an essential procedure for preserving and inferring the latent properties of the population.

A sampled graph is representative if it preserves selected properties of the original graph. In present studies, the simple or explicit characteristics/properties, such as degree, clustering-coefficient and shortest-path, are usually of interest to data scientists, but these properties cannot fully reflect intrinsic topological properties (e.g., clustering structure) of graphs. Also, in the current research, sampling algorithms [8,9,13,15,18,20] mostly study static graphs and suppose that the graphs can be retrieved as a whole through multiple passes. Nevertheless, contemporary graphs should be abstracted into fully-dynamic streaming graphs in which the edge additions and deletions occur arbitrarily at any time [26]. Analyzing such graphs has become of great importance to investigate the structural properties and evolution of graphs in various domains (e.g., detecting ransomware attacks and abnormal connections in social media). Therefore, online sampling from streaming graphs is preferred to sample representative subgraphs as new entities arrive in a form of stream, instead of static graph snapshots. Note that in many domains, graph mining is rapidly moving from analyzing static snapshots to ever-increasing streaming graphs [4]. Therefore there are two research challenges that should be resolved:

One is to develop an efficient and effective online sampling algorithm on fully-dynamic streaming graphs. In particular, the sampling method should be capable of processing the edge insertions and deletions in an incremental manner, where each edge should be processed only once and extremely fast using the limited memory.

The other is to produce representative samples from the unbounded, ever-increasing streaming graphs. The sample is a representative subgraph if it retains the original graph's properties. Maiya et al. [19] argue that the representativeness should be consistent with the properties that need to be preserved. Most of sampling approaches only assess the extent to which the sampled subgraph is representative of explicit/simple topological properties (e.g., the degree distribution), but are not adequate for retaining the intrinsic property, i.e., clustering structure, which is prevailing in many real-world graphs under study today [1,10,33,35].

To tackle these challenges, we formulate the problem of sampling from fully-dynamic streaming graphs, and the main contribution can be summarised as follows:

- We propose a *C*luster-preserving *P*artially *I*nduced *E*dge *S*ampling (*CPIES*) algorithm to process fully-dynamic streaming graphs in which edge insertions and deletions are allowed. It employs: (i) the cluster-preserving node replacement strategy, (ii) the elimination method of isolated nodes, and (iii) the edge deletion operation. It makes samples capable of representing clustering structure at any time, and completely avoids isolated nodes in the sampled subgraph. Furthermore, it supports the edge deletion operations which is of crucial importance in a fully-dynamic streaming setting.
- We present a new two-stage inference method to infer nodes' clustering affiliations in the original streaming graph (i.e., population). If the sample is a good cluster-preserving representative of the original population, the clustering structure of the sample should generalize well to the nodes that are not sampled. It provides a feasible solution to infer clustering affiliations of nodes in the population.
- Experiments are conducted to demonstrate that the proposed *CPIES* algorithm is capable of preserving the clustering structure while keeping an up-to-date sample of fully-dynamic streaming graphs. It is superior to competing sampling algorithms in most properties, especially in terms of clustering performance. Furthermore, the two-stage clustering inference (*TCI*) method in conjunction with *CPIES* provides a feasible solution to inference clustering affiliations of the population.

The paper is organized as follows. In Section 2 we overview the related work on graph sampling and formalize the problem statement in Section 3. The proposed *CPIES* and *TCI* algorithms are elaborated in Sections 4 and 5, respectively. The results of empirical study are discussed in Section 6. We summarize the work and outlook the future work in Section 7.

2. Background & related work

Graph sampling is of interest to researchers in a wide range of domains. Although each domain may investigate different type of graphs, the focus has been on designing representative sampling algorithms from the large population, and utilizing the subgraphs to simplify the downstream task (e.g., counting triangle [21], classification [1] and clustering [30]). The researches related to this work can be grouped into: (a) streaming graph sampling; (b) clustering structure in graphs.

2.1. Streaming graph sampling

Traditionally, graph sampling has been well-studied in respect of static graphs. It is supposed that the graphs have static structure and can fit into the memory, thus sampling algorithms can explore any node and the full neighborhood of each node at any step. Also, the temporal evolution property is ignored. However, the graphs encountered into real-world applications are massive-size and/or in a streaming fashion. Since such graphs are quite ubiquitous nowadays, an increasing number of studies focus on developing sampling algorithms that address the complexities of streaming model. There are mainly three aspects that streaming graph is diverse from static graphs: (i) the structure/characteristic cannot be completely measurable (e.g., only support a sequential access); (ii) the size of streaming graph is open-ended such that it cannot fit into the restricted memory; and (iii) accurate and efficient processing is of great importance [5,32]. Thus, efficient online

sampling methods on such streaming graphs should be devised to obtain representative subgraphs which reflect the properties of original streaming graphs perfectly.

Ahmed et al. [3] put forward a streaming graph sampling framework, namely *Graph Sample and Hold* (*gSH*), to handle insert-only streaming graphs. It supports to sample sequentially from streaming graphs by one pass under the limited memory footprint [5]. converted the *edge-based sampling, node-based sampling* and *breadth-first sampling* of static graphs into those of streaming settings, and proposed a *partially induced edge sampling* (*PIES*) algorithm to handle streaming graphs. *PIES* inserts the first n nodes into the *node reservoir* and then randomly replaces existing nodes afterwards by newly arrival ones. For each new edge, it will be added to the sampled graph if its two incident nodes are in the *node reservoir*. Once a sampled node is substituted, all its incident edges are dismissed accordingly from the sampled graph. The sampling procedure continues until the edge stream terminates. *PIES* focuses on uniformly sampling from streaming graphs using a single pass but has no guarantee on the sample quality in terms of preserving the clustering structure. Moreover, edge deletion requests are not supported in it, thus it cannot be applied to the fully-dynamic streaming setting, for instance when sampling is executed over a sliding window (i.e., out-dated edges should be erased from the end of sliding window). A graph priority sampling (*GPS*) proposed in [4] is a variant of *reservoir sampling* from streaming graphs. It proposed a general approach to sample weighted edges to achieve specific estimation goals of graph properties on the basis of additional variable. However, how to sample the clustering structure is still not resolved in this method.

Some other related work mainly focused on graph mining problems such as counting of triangles [21], estimation of pagerank [25], outlier detection [1], link prediction [23,36] on streaming graphs. Our objective is diverse from those previous work due to existing sampling techniques are inadequate for the essential property: inherent clustering structure. Thus the aim of this paper is to achieve sampling on fully-dynamic streaming graphs as well as to retain the inherent clustering structure.

2.2. Clustering structure in graphs

Blondel et al. [6] defined a cluster as a set of nodes which have close interactions with each other and sparse connection to nodes outside of the cluster. The clustering structure is intrinsic and prevalent property in real-world graphs, and usually corresponds to the nodes's latent properties such as affiliation, research interest and social role [34]. Though sampling focuses on representing the latent properties embedded in the original population, most sampling methods are not particularly devised to retain the clustering structure. In the scope of our knowledge, sampling methods have not been employed to retain the clustering structure in fully-dynamic streaming graphs. However, the present studies have demonstrated that contemporary social networks tend to exhibit explicit clustering structures [14,27] and increasing density over time [15,16].

The work in [15] defined several scale-down and back-in-time properties that the sample should represent, and evaluated a set of sampling techniques in preserving these properties of static graphs. Ebbes et al. [9] also assessed several sampling methods' performances in preserving four primary properties (i.e., closeness-centrality, degree, clustering-coefficient and betweenness-centrality). However, the representativeness of the sample is evaluated by these explicit properties that cannot fully represent the topology structure. In [19] two sampling algorithms were proposed to sample representative subgraphs from the perspective of the clustering structure. Both methods are extended from the concept of expander graph. The subgraph with higher expansion rate should be more representative of the clustering structure, so newly sample nodes can be chosen into the sample either deterministically or probabilistically using two different strategies. However, their drawback is that they can only handle static graphs to retain the clustering structure.

3. Problem statement

In this work, our aim is to sample the clustering structure from fully-dynamic streaming graphs in which edge insertions and deletions are supported. The better the clustering structure is retained in the sample, the better the representativeness of samples is.

Firstly, we give the definition of fully-dynamic streaming graph. Formally, for any discrete time-stamp $t \geq 0$, a graph $G(t) = [V(t), E(t)]$ can be treated as input which is presented as an edge-stream $E(t)$, where $V(t)$ is a finite set of nodes and $E(t) \subseteq V(t) \times V(t)$ is a set of edges by time t. Each edge $e(t)$ is in the form of $\langle u, v, t \rangle$, where u and v are the two incident nodes of the edge and t is the associated time-stamp. Initially when time-stamp $t = 0$, $V(t) = E(t) = \emptyset$. Subsequently for each discrete time-stamp t $(t > 0)$ a new update $e_t = (\bullet, \langle u, v, t \rangle)$ where $\bullet \in \{+, -\}$ is received from the edge streams. We update the graph $G(t) = [V(t), E(t)]$ by time-stamp t as follows:

$$E(t) = \begin{cases} E(t-1) \cup \langle u, v, t \rangle & \text{if } \bullet = + \\ E(t-1) \setminus \langle u, v, t \rangle & \text{if } \bullet = - \end{cases} \tag{1}$$

Note that the edges in the graph is undirected and the same edge can appear multiple times in the stream.

The task we focus on is to sample representative subgraphs from streaming graphs that should obtain a good quality of sampling. Formally, $\eta(\cdot)$ is denoted as any graph property/characteristic and it could be a single-valued statistic (e.g., average clustering-coefficient) or a multiple-valued distribution (e.g., k-core distribution) [3]. The goal is to guarantee that subgraph $G_s(t) = [V_s(t), E_s(t)]$ is representative, which means that it should preserve selected graph properties of G, i.e.,

$\eta(G(t)) \approx \eta(G_s(t))$. In this work, we particularly take the inherent clustering structure into account. Apart from the requirement for the sampling representativeness, it is also requested to process the streaming graph by single-pass under the limited memory χ. The formal definition is as follows.

Definition 1. Streaming graph sampling. given the maximum memory size χ and a streaming graph $G(t)$, the goal of the sampling algorithm $\mathcal{S}$ is to generate a representative subgraph $G_s(t)$ by sequentially sampling edges from $E(t)$ such that:

- the memory required to store the subgraph $G_s(t)$) (i.e, $\mathcal{O}(|G_s(t)|)$) should satisfy the maximum memory size, i.e., $\mathcal{O}(|G_s(t)|) \leq \chi$.
- the edges are updated through a single pass;
- the graph properties, especially the clustering structure, should be representative and preserved well, i.e., $\eta(G(t)) \approx \eta(G_s(t))$.

What do we mean by "clustering structure is preserved" in fully-dynamic streaming graphs? First, the sampled counterpart should contain influential/hub nodes from most (or even all) clusters of the original streaming graphs (i.e., a stratified sample of clustering structure). Second, the nodes grouped together in the sampled counterpart should come from nodes that locate in the same cluster of the original streaming graph [19]. We give the formal definition as follows:

Algorithm 1 Basic PIES algorithm.

Input:
Streaming graph by time-stamp t: $G(t) = (V, E)$;
Sample size: n.
Output:
Sampled subgraph by time-stamp t: $G_s(t) = (V_s, E_s)$.
$V_s \leftarrow \emptyset, E_s \leftarrow \emptyset, t \leftarrow 0$
while the edge stream e_t arrives at t **do**
  $e_t = (u, v, t)$
  ###{**Initialization part**}
  if $|V_s| < n$ **then**
    if $u \notin V_s$ **then**
      $V_s \leftarrow V_s \cup \{u\}$
    end if
    if $v \notin V_s$ **then**
      $V_s \leftarrow V_s \cup \{v\}$
    end if
    $E_s \leftarrow E_s \cup \{e_t\}$
    $m \leftarrow |E_s|$
  else
    ###{**Updating part**}
    $p_e \leftarrow \frac{m}{t}$
    $r \leftarrow Random(0, 1)$
    if $r < p_e$ **then**
      if $u \notin V_s$ **then**
        u'=Select_Replaced_Node(V_s)
        $V_s \leftarrow V_s \cup \{u\} - \{u'\}$
        for each edge e' incident to u' in E_s **do**
          $E_s \leftarrow E_s - \{e'\}$
        end for
      end if
      if $v \notin V_s$ **then**
        v'=Select_Replaced_Node(V_s)
        $V_s \leftarrow V_s \cup \{v\} - \{v'\}$
        for each edge e' incident to v' in E_s **do**
          $E_s \leftarrow E_s - \{e'\}$
        end for
      end if
    end if
    if $u, v \in V_s$ **then**
      $E_s \leftarrow E_s \cup \{e_t\}$
    end if
  end if
  $t \leftarrow t + 1$
end while
Return: G_s

Definition 2. Given a streaming graph $G(t)$, a clustering algorithm $\mathcal{P}$, and a measurement of clustering similarity $\mathcal{M}[\bullet, \bullet]$, a subgraph $G_s(t)$ is a cluster-preserving sample that satisfies the following two conditions at timestamp t:

- $G_s(t)$ maximizes the clustering similarity between the identified clusters $\pi(G_s(t))$ produced by $\mathcal{P}$ on $G_s(t)$ and the clusters $\pi(G(t))$ produced by $\mathcal{P}$ on $G(t)$, i.e., $\max \mathcal{M}(\pi(G(t)), \pi(G_s(t)))$.
- the intersection set between sampled node-set $V_s(t)$ and each cluster of $\pi(G(t))$ is maximized, where $\pi(G(t))$ is the set of clusters that are produced by $\mathcal{P}$ on $G(t)$.

Note that we sometimes drop t from the notation if it is clear from the context. Based on the above definitions, we present a novel cluster-preserving streaming sampling algorithm which is capable of preserving the clustering structure of the original streaming graph well.

4. Proposed sampling method

In this section, first of all, the state-of-the-art *PIES* (Partially Induced Edge Sampling) algorithm [5] is introduced. Secondly, the proposed *CPIES* algorithm is detailed introduced and it employs: (i) the cluster-preserving node replacement strategy, (ii) the elimination method of isolated nodes, and (iii) the edge deletion operation to handle the fully-dynamic streaming graph.

4.1. The basic PIES algorithm

Essentially, *PIES* [5] is a two-phase streaming graph sampling algorithm through a single pass: Initially, initial edges in the stream are added to the *edge reservoir* deterministically in order to accumulate the *node reservoir* V_s. When the size of $|V_s|$ is reached to n, the number of sampled edges in the *edge reservoir* E_s is denoted by m. Then the sample's update comprises two phases to handle the new edge e_t at time-stamp t:

- *Selection phase*: each new edge e_t is assigned a sample probability of $p = \frac{m}{t}$ where t is the time-stamp of e_t. If the new edge meets the probability, each incident node of e_t is added into the *node reservoir* and the process goes to replacement phase. Otherwise, the new edge e_t is not sampled. The rationale is inherit from reservoir sampling principle [28] which makes each edge in $E(t)$ at timestamp t have the equal probability $\frac{m}{t}$ to be sampled.
- *Replacement phase*: if the new edge is chosen and no less than one associated node has not been sampled into V_s, the previously sampled nodes in the *node reservoir* V_s are substituted according to a specific strategy (namely, *Select_Replaced_Node*$(\cdot)$) to keep the target capacity of V_s.

Subsequently, the new edge e_t is included into the *edge reservoir* E_s if its two associated nodes have been chosen into the *node reservoir* V_s (i.e., partial edge induction). Through the investigation, we conclude that the node replacement strategy *Select_Replaced_Node*$(\cdot)$ should vary according to different requirements and accept various node replacement strategies to select the node which needs to be replaced (namely, replaceable node) [5]. propounds two replacement strategies for *PIES*. One is to uniformly select the replaceable node at random from the *node reservoir* V_s, whereas the other is to substitute the node kept in V_s for the longest time without gaining more edges and has minimum degree which is denoted by *PIES (Min)*. Obviously, the latter strategy contributes more to the quality of samples (e.g., less isolated nodes), but it is more costly and inefficient since it needs to maintain and traverse a list recording the recent update time of each node in V_s when an edge arrives. Nevertheless, both of replacement strategies do not consider to retain the inherent structure as their main goal.

4.2. Cluster-preserving node replacement

As illustrated above, when sample size n is reached, upcoming edges and their incident nodes should be probabilistically sampled and replace the sampled nodes from the *node reservoir* V_s. Although the replacement strategy of *PIES (Min)* may keep nodes with high degrees and also remove some of out-dated nodes, it is not sufficient to sample the clustering structure because: (i) hub nodes in small clusters usually have more influence than ordinary nodes in the same cluster, but have less degrees than some ordinary nodes in larger clusters. This strategy might drop these hub nodes in small clusters such that the clustering structure in the sample is not complete; (ii) with the new edges rolling on, the replaceable nodes in this strategy may not have new updates for a long time and keep losing influence on its cluster, but it still can be a hub node in its cluster before other nodes have enough power (e.g., high degree) to replace its central position in the streaming process. For instance, when an old leader of a political party is retiring and a new leader is becoming "rising star". In this process, the new leader will probably become more and more central in the party, while the old one will keep losing his refluence. However, the old leader should not be replaced until the new leader has more influence on the party.

Since current replacement strategies do not consider the clustering structure into account, we propose a new cluster-preserving replacement strategy to retain the hub nodes and replace a fraction of less-important nodes in the sample to preserve the inherent structure. The basic rationale is that the status of a node in a cluster is subject to its degree, and the hub node tends to possess higher degree than that of its neighbors. Therefore if a node to be substituted is of high degree, we deem it has higher chance to be the hub node of its cluster such that the sampling method should further confirm whether it should be selected as the replaceable node.

As described in Algorithm 2, u' is the selected node uniformly at random, $N_s(u')$ is a set of neighbor nodes of u' in V_s, and *flag* is a boolean variable used to record if the selected node u' has higher degree than that of all its neighbors. The smallest degree of neighbour $N_s(u')$ of node u' is denoted by *min*. First, we uniformly select the node u' that could be replaceable at random. Second, our strategy is to check whether to substitute the node or its neighbour. The workflow is as follows: it compares the degree of node u' with that of all its neighbors. If the node's degree is higher than all its neighbors', which indicates the node is probably still a hub node in the cluster, we should not replace the current selected node u' but one of its neighbor with the lowest degree as replaceable node instead [22]. In this way, hub nodes of the sample should be maintained on the fly, therefore we preserve the inherent clustering structure of original streaming population.

Note that the new replacement strategy can be generalized into higher-order definition of hub nodes in the cluster, The proposed *CPIES* method considers the nodes which have more degrees (i.e., 1-hop neighbors) as hub nodes. However, we can extend the 1-hop neighbours to m-hop neighbors to determine the hub nodes and less-important nodes in the clusters. However, higher order m-hop ($m > 1$) extension suffers from two limitations: (1) from *computational* perspective it is more time-consuming since m-hop neighbour information for each node need to be calculated and retrieved; (2) from *experimental* perspective, it has the similar performance with *CPIES*. Due to the similar results, we would not report experimental results of high-order extension.

Algorithm 2 Proposed Select_Replaced_Node(V_s).

Require: Set of sampled nodes: V_s;
Ensure: Selected node to be replaced: u''.

$i \leftarrow discreteUniform[1, |V_s|]$
$u' \leftarrow V_s[i]$
$flag \leftarrow 1, u'' \leftarrow u', \min \leftarrow |N_s(u')|$
for all $ne \in N_s(u')$ **do**
  if $|N_s(ne)| > |N_s(u')|$ **then**
    $flag \leftarrow 0$
    break
  else
    if $|N_s(ne)| < \min$ **then**
      $min \leftarrow |N_s(ne)|$
      $u'' \leftarrow ne$
    end if
  end if
end for
if $flag = 0$ **then**
  $u'' \leftarrow u'$
end if
return u''

4.3. Isolated nodes elimination

We denote *isolated nodes* by those nodes kept in the *node reservoir* V_s with no edges attached. There are two reasons that *isolated nodes* exist in the sample in *PIES*.

(1) In replacement phase, random selection of the replaceable node may make the newly added node become isolated. Here a concrete case is illustrated to show how such kind of isolated nodes is yielded.

Example 1. we suppose that at the timestamp t the new edge $e_t = (u, v, t)$ is arriving and need to be sampled into E_s when the *node reservior* $V_s = [a_1, a_2, \ldots a_n, u]$ (i.e., u belongs to V_s but v does not). Since V_s does not contain the node v, *PIES* need to choose a replaceable node for it randomly. However, node u has a probability $p = 1/|V_s|$ to be picked to be the replaceable one and it will be erased from V_s when node v is added. If u is chosen, u will be deleted and not belong to V_s any more right after the node replacement, the new edge e_t will not be sampled such that v becomes an isolated node.

Clearly, this situation happens from time to time since an incident node of new edge is selected as the one to be replaced. To resolve this problem, a stack structure, denoted by *ForbiddenStack*, is designed and we stipulate that each node located in *ForbiddenStack* cannot be chosen as the replaceable node. Therefore, for each new edge, its incident nodes should be pushed into *ForbiddenStack*. That is, they are prohibited to be selected as replaceable nodes. When the next edge arrives, nodes in *ForbiddenStack* should be popped up such that it would not impact the node replacement in the new update.

(2) After the replaceable node needs to be replaced, some neighbour nodes only have a link with the replaceable node and they will become isolated nodes. The reason is that *PIES* eliminates associated edges of the replaceable node such that the neighbours which only have connection with the replaceable node will have no edge attached.

Our strategy is to check whether the neighbours' degrees of the replaceable node fall to zero. If so, it means those neighbours become isolated nodes and need to be eliminated. Note that the strategy of *PIES (Min)* is helpful to get rid of such kind of nodes. However, the high cost to maintain and traverse a time list of each node in sample process makes it impractical for open-ended streaming graphs and not compatible with new replacement strategies. On the contrary, the proposed isolated nodes elimination is simple yet efficient and compatible well with new replacement strategies, which makes it more versatile and feasible in fully-dynamic streaming graphs.

4.4. Edge-deleting operation

Essentially, supporting the edge-deletion request is of great importance to handle fully-dynamic streaming graphs. For instance, obsolete edges should be eliminated from the sliding window in a timely manner if sampling is executed over a sliding window. However, it is not taken into consideration by *PIES*. Therefore, an efficient delete method is proposed to support edge-deletion requests. The description is depicted in Algorithm 3. Once an edge-deletion is demanded, the algorithm checks whether the corresponding edge exists in E_s. If so, we remove the edge from the *edge reservoir* directly.

Algorithm 3 Edge-deletion(e_t, G_s).

Require:
Edge to be deleted: $e_t = (-, \langle u, v, t \rangle)$;
Sampled subgraph: $G_s = (V_s, E_s)$.
Ensure:
Updated sampled subgraph: G_s.
if $e_t \in E_s$ **then**
  $E_s \leftarrow E_s - \{e_t\}$
  if $|N_s(u)| = 0$ **then**
    $V_s \leftarrow V_s - \{u\}$
  end if
  if $|N_s(v)| = 0$ **then**
    $V_s \leftarrow V_s - \{v\}$
  end if
end if
return G_s

Otherwise, we omit this request which means the corresponding edge has not been sampled before. Since an edge-deletion request implies that the edge's influence on graph's structure should be eliminated instantly, it makes sense that the edge-deletion requests should be met all the times. Furthermore, we examine the degrees of associated nodes in the deleted edge and remove them if their degree falls to zero after the edge-deletion operation [22].

4.5. Cluster-preserving partially induced edge sampling (CPIES)

According to these extensions, a cluster-preserving partially induced edge sampling algorithm (*CPIES*) is proposed to process the fully-dynamic streaming graph. The overall description is depicted in Algorithm 4.

The proposed *CPIES* is expected to keep hub nodes of each cluster into the sample and eliminate the isolated nodes which are meaningless for any topological property. Thus, it is capable of preserving the clustering structure in the samples considerably.

5. Inferring clustering affiliation from samples

Since graph sampling is a core part of population inference, we need to recover the clustering affiliation of nodes for all the unsampled nodes in the population. For this purpose, we propose a new *two-stage clustering inference* (*TCI*) method to label the un-sampled nodes. The rationale is that if the sampled graph is a cluster-preserving subgraph of the original population, the clustering structure on the sample should generalize well to the nodes that are not sampled. That is, using a subgraph G_s at any time-stamp t, our target is to infer the clustering affiliation for each unsampled node v where $v \in V - V_s$. Since low-degree nodes are not vital for inherent clustering structure, the clustering inference method is composed of two stages: 1) initialization of clustering labels for unsampled nodes; 2) label propagation for the whole graph.

As algorithm 5 shown, in the coarse screening stage, first we sort the unlabeled nodes in a descending order according to the ratio of their labeled neighbors such that the label information of samples are sufficiently utilized. Second for each unlabeled node from the sorted queue U, we assign clustering labels of the unlabeled node to the label of its maximum occurrence among the neighbours. Then we remove it from the unlabeled set U and add the node into the labeled node-set V_s until no more nodes in V can be labeled. For the isolated nodes which are not connected to the sample, they are designated into its own single-node cluster.

In the fine adjustment stage, we utilize *label propagation* (*LP*) clustering algorithm [24] to optimize the clustering structure for the population at any time. *LP* algorithm iteratively simulates a process in which each node in the graph picks the most frequent label in its neighbourhood in each iteration. The process repeats until all the nodes reach a consensus. The advantages of this approach is that (i) it does not need a pre-defined objective function; (ii) its time complexity increases linearly with the size of the graph, i.e., $\mathcal{O}(m+n)$; (iii) it has the ability to set the initial labels and also allows some labels to be fixed. Thus we set the labeling results of the coarse stage to the initial labels, and assign the clustering labels of the sampled nodes to fixed labels whose labeling should not change during the iteration. In this way, the *TCI* method is able to assign and infer the clustering labels to the population effectively and accurately.

6. Experimental evaluation

In this section, first, we briefly introduce the datasets we employed. Then we define a suite of graph properties and give the evaluation methodology. Finally, extensive experiments are conducted and some remarkable results are elaborated and discussed.

Algorithm 4 Cluster-Preserving PIES Algorithm.

Require:
Streaming graph by time-stamp t: $G(t) = (V, E)$;
Sample size: n.
Ensure:
Sampled subgraph by time-stamp t: $G_s(t) = (V_s, E_s)$.

$V_s \leftarrow \emptyset, E_s \leftarrow \emptyset, t \leftarrow 0$
while the edge stream $(symbol, e_t)$ arrives at t **do**
  $e_t = (u, v, t)$
  ###{**Edge addition**}
  if symbol = '+' **then**
    if $|V_s| < n$ **then**
      if $u \notin V_s$ **then**
        $V_s \leftarrow V_s \cup \{u\}$
      end if
      if $v \notin V_s$ **then**
        $V_s \leftarrow V_s \cup \{v\}$
      end if
      $E_s \leftarrow E_s \cup \{e_t\}$
      $m \leftarrow |E_s|$
    else
      $p_e \leftarrow \frac{m}{t}$
      $r \leftarrow Random(0, 1)$
      if $r < p_e$ **then**
        $ForbiddenStack \leftarrow ForbiddenStack \cup \{u, v\}$
        if $u \notin V_s$ **then**
          $u' \leftarrow Select\_Replaced\_Node(V_s)$
          $V_s \leftarrow V_s \cup \{u\} - u'$
          for each edge $e' = (u', x)$ incident to u' in E_s **do**
            $E_s \leftarrow E_s - \{e'\}$
            if $|N(x)| = 0$ **then**
              $V_s \leftarrow V_S - \{x\}$
            end if
          end for
        end if
        if $v \notin V_s$ **then**
          $u' \leftarrow Select\_Replaced\_Node(V_s)$
          $V_s \leftarrow V_s \cup \{v\} - v'$
          for each edge $e' = (v', y)$ incident to v' in E_s **do**
            $E_s \leftarrow E_s - \{e'\}$
            if $|N(y)| = 0$ **then**
              $V_s \leftarrow V_S - \{y\}$
            end if
          end for
        end if
        $ForbiddenStack \leftarrow ForbiddenStack - \{u, v\}$
      end if
      ###{**Partial edge induction**}
      $E_s \leftarrow E_s \cup \{e_t\}$
    end if
  end if
  ###{**Edge deletion**}
  if symbol = '-' **then**
    $G_s \leftarrow Edge\text{-}deletion(e_t, G_s)$
  end if
  t++
end while

Algorithm 5 Inferring Clustering Affiliation from Samples.

Require:

Streaming graph by time-stamp t: $G(t) = (V, E)$;
Sampled subgraph by time-stamp t: $G_s(t) = (V_s, E_s)$;
Clustering labels of the sampled nodes V_s: l_{V_s}

Ensure:

Clustering labels of $V(t)$ in $G(t)$: l_V.

Obtain the unsampled nodes: $U \leftarrow V - V_s$
###{**The coarse screening**}
Sort the nodes of the node-set U according to the fraction of the neighbours in V_s
while no more nodes in U can be labeled **do**
  for each node $u \in U$ that pops up from U **do**
    ###{Set its label to the maximum occurring label among the neighbours in V_s}
    $l_V(u) \leftarrow \arg\max_{l_{V_s}}[Count(l_{V_s}(ne))]$ s.t. $ne \in N_s(u)$,
    $U = U - u$
    $V_s = V_s \cup u$
  end for
end while
if $u \in U$ is a isolated node **then**
  $l_V(u) \leftarrow$ single-node label
end if
###{**The fine adjustment**}
Set the initial node labels: $initial = l_V$
Set the fixed labels whose labels should not change: $fixed = l_{V_s}$
$l_V \leftarrow$ Label_propagation(G(t), initial, fixed)
return l_V

Table 1
The statistical information of streaming graphs. Corresponding snapshots are obtained by accumulating streaming graphs according to the specific time interval and the end streams' statistics are given. Abbreviations are as follows: *# snapshots*: the number of snapshots; *# simple edges*: the number of non-loop and non-duplicate edges; *# comps*: number of connected components; *CC*: the average clustering-coefficient for all the nodes. Please note that "*" means it is unknown in synthetic graphs.

G	$\|V\|$	$\|E\|$	# simple edges	Time span	Interval	#snapshots	Statistics of the end streams				
	(Maximum comps)			(Days)	(Months)		#Comps	Diameter	Radius	Density	CC
MS_1000	1000	4442	4442	*	*	5	1	6	5	0.00889	0.082
EC_1000	1000	4304	4304	*	*	5	1	6	5	0.0092	0.086
Enron-employee	151 (150)	50,572	1611	1138	6	7	2	4	3	0.14400	0.5210
Email-Eu-core	986	332,334	16,064	803	2	14	1	7	4	0.03308	0.4070
CollageMsg	1899 (1893)	59,835	13,835	193	1	7	4	8	4	0.00772	0.1097
Reality	6809	52,050	7697	106	0.5	8	1	8	4	0.000332	0.0178
Reply-Slashdot	51,068	280,443	117,340	371	2	7	1	17	9	0.00009	0.0201
Fackbook-wall	46,952 (43,953)	876,933	182,384	1560	4	14	842	18	9	0.00019	0.1149

6.1. Experimental setup

The experiments are carried out on a Linux server runing CentOS. Each run employs a single core with 2.40 GHz CPU and at most 32 GB main memory. The proposed *CPIES* and competing methods (i.e., *PIES, PIES (Min), StreamES, StreamNS*) are all implemented by C++[1]. For each sample rate p, five trials are run in each experiment and multiple metrics for each designated snapshot are obtained in benchmark graphs.

6.2. Datesets

Synthetic and real-world graphs from a variety of domains are used to verify the effectiveness of the proposed *CPIES* algorithm. A brief summary of these graphs is shown in Table 1.

[1] The source codes and default parameters of the implementations of all algorithms can be found at https://github.com/feilong0309/Streaming_graph_sampling.

6.2.1. Synthetic graphs

We adapt the dynamic network benchmark [12] to generate a set of step graphs with embedded clustering structures. The shifting between step graphs are manipulated by the injection of a user-specified number of evolving events, e.g., merging/splitting or birth/death. These step graphs share similar characteristics and have the ground-truth embedded in them. To produce the streaming edges, we record the generation orders of edges as their timestamps and rearrange the edges randomly in each trial. We ensure that all methods use the same sequential order in such a way that the quantitative comparison can be given along with edge updates.

In our implementation we generate two synthetic graphs for two different event types, including merging/splitting and expansion/contraction, over five time steps. Meanwhile, at each subsequent timestep, a number of instances of corresponding events occur. Edge updates are streamed in the system according to their sequential order and all the metrics are calculated at the specific five snapshots in which the ground-truth is known. The step graphs share a number of parameters according to [12] to simulate real-world graphs: the number of nodes: $N = 1000$, the maximum degree: $maxD = 16$, the average degree: $D = 10$, the average ratio of external degree to total degree for each node: $\mu = 0.2$, and the minimum and maximum cluster size: $maxC = 40$, $minC = 60$, respectively. Subsequently, two synthetic benchmarks are generated as follows:

- Merging/splitting (*MS*) benchmark: the instances where merging and splitting events are embedded in the stream. Initially, we obtain the initial clusters. In the subsequent snapshots, 2 instances of cluster splittings occur, together with 2 instances of existing clusters are merged.
- Expansion/contraction (*EC*) benchmark: we create rapid cluster expansion and contraction in each step graph where 2 selected clusters randomly contract or expand by 25% of the previous size. In terms of cluster-expansion, the nodes of new clusters are chosen at random from other clusters.

6.2.2. Real-world graphs

Real-world graphs are chosen from different domains (e.g., scholar citations, social media, web-crawling). To normalize the timescales in different graphs, we use a sequence of integers as associated timestamps. They can be downloaded from the Stanford Large Network Dataset Collection [17], and brief descriptions are listed as follows:

- *Enron-employee*: It covers email communication from 1999 to 2003 within 151 Enron employees. Nodes represent core employees and edges are email communications among them. Note that this graph might contain self-loop edges.
- *Email-Eu-core*: It is generated from a large European research institution. The e-mails only represent communication among core members in the institution. The edge (u, v, t) denotes that user u has an email communication with user v at time-stamp t.
- *CollegeMsg*: It consists of an anonymous social network's messages at a US University. Users could seek for others and then start conversations. The edge connection means that users have a private message between them.
- *Facebook-wall*. It originates from the wall posts among users on Facebook around New Orleans. An edge (u, v, t) implies that user u posted on facebook wall of user v at time t.
- *Reply-Slashdot*: Nodes are users of the Slashdot website. Edges are the responses among users and attached with timestamps of the responses.
- *Reality*: It consists of mobile phone call events among a group of users at MIT. A user is represented by the node and voice mail or a phone call is indicated by an edge between them.

6.3. Evaluation measures

6.3.1. Basic properties & measurements

Basic properties. They can be roughly divided into two categories: point-statistic (i.e. single-valued statistic) and distribution (i.e., multi-valued statistic). Obviously, a wide range of graph properties is of importance to investigate how is the behaviour of sampling on the population. Table 2 summarizes graph properties we employed. Besides, we formally give the definition of various distributions as follows:

(1) *Degree distribution*: degree centrality ($deg(u)$) indicates the number of neighbour nodes adjacent to the node u, thus the degree distribution is defined as follows:

$$\forall d > 0, \quad p(deg(u) = d) = \frac{|\{u \in V | deg(u) = d\}|}{|V|}.$$

It is used to describe the distribution of nodes with different degree over the graph, and it is an essential property to analyze the node importance in graphs.

(2) *Shortest-path distribution*: it is defined as the ratio of node-pairs $(u, v) \in V$ whose shortest-path ($dist(u, v)$) is equal to s. Formally, it can be defined as follows:

$$\forall s > 0 \ \& \ s \neq \infty, \quad p(dist(u, v) = s) = \frac{|\{(u, v) \in V | \ dist(u, v) = s\}|}{|V|^2}.$$

It reflects the distance distribution of each pair of nodes over all the distances. Note that the shortest-path is only calculated in the connected component, thus the maximum component should be considered when needed.

Table 2
Graph statistics.

Types	Statistics	Description
Point-statistics	# of edges	Number of edges in the graph
	# of clusters	Number of clusters in the graph
	Diameter	The greatest distance among nodes in the graph
	Radius	The minimum eccentricity of nodes in the graph
	Density	The proportion of the actual number of edges to all possible edges
	Average CC	The mean value of the clustering-coefficient for all nodes
	Modularity	The modularity of the graph
Distributions	Degree	Degree distribution of nodes
	Shortest-path	Shortest-path distribution of each pair of nodes
	CC	Clustering-coefficient distribution for all nodes
	K-core	K-core decomposition's distribution

(3) *Clustering-coefficient distribution*: the clustering-coefficient is defined as the ratio of the actual number of edges between the node's neighbors to the maximum possible number of edges between them. Therefore, clustering-coefficient distribution of the whole graph denotes the proportion of nodes whose clustering-coefficient ($cc(v)$) is equal to c ($c \geq 0$). Formally, we define it as follow:

$$\forall c > 0, \quad p(cc(u) = c) = \frac{|\{u \in V|\ cc(u) = c\}|}{|V|}.$$

In general, the larger the clustering coefficient is, the more interconnections each node has. In real-world graphs, nodes tend to establish tightly-knit clusters by relatively dense connections. Therefore it is an effective metric to measure the degree to which nodes are inclined to cluster together.

(4) *K-core distribution*: the induced subgraph with minimum degree k is denoted by k-core of G. That is, given a subgraph $G_s = (V_s, E_s)$ where $E_s = \{e_{\{u,v\}} \in E \mid u, v \in V_s\}$, G_s is a k-core subgraph with minimum degree k if it satisfies that $\forall v \in V_s$ *s.t.* $deg(V_s(v)) \geq k$.
Thus, the *k-core distribution* is considered as the proportion of nodes which involve in k-core decomposition. It is crucial to analyze vulnerability of the network since it exhibits the clustering structure and connectivity of the graph.

Measurements of basic property. A ood?sample should be representative in a sense that graph properties of interest can be preserved with high quality guarantee. Thus, the distance measurement between the property in G and the one in G_s (i.e., $dist[\eta(G), \eta(G_s)]$) is widely used to evaluate sample representativeness quantitatively, and sampling algorithms that minimize the distance are considered superior. Here, $\eta(\cdot)$ represents the distribution of a specific network property (e.g., degree distribution). The distribution reflects how the graph structure is distributed across nodes and edges. In this study, we employ two distributional distance measures, i.e., Jensen-Shannon (JS) and Kolmogorov-Smirnov (KS) divergence, to measure the discrepancy between the distribution of original population and that of the sample. Smaller value of divergence indicates better fitness between two distribution functions.

(1) Kolmogorov–Smirnov (KS) divergence: it is employed to compute the divergence between the distributions of two *Cumulative Distribution Functions (CDFs)*. F_H and F_K represent the CDFs of the population and sample, respectively. Let x denote the random variable, it is the maximal perpendicular distance between the two CDFs, i.e.,

$$KS(H, K) = max_x|F_H(x) - F_K(x)|.$$

(2) Jensen-Shannon (JS) divergence: it is a measure of how one probability distribution diverges from an expected probability distribution. It inherits from Kullback-Leibler (KL) divergence (namely, relative entropy) and is robust with isolated nodes and histogram bins. It can be calculated as follows:

$$JS(H, K) = 1/2 \sum_i (h_i \log_2(\frac{h_i}{m_i}) + k_i \log_2(\frac{k_i}{m_i})),$$

where $m_i = \frac{(h_i + k_i)}{2}$, H and K are *probability density function (PDF)* distributions of the population and the sample. h_i and k_i are the relative frequencies attached to H and K, respectively.

In our experiments, we sample 20% of total number of nodes from each graph by using multiple sampling methods. Afterwards, we use KS and JS divergences to measure the differences of *PDF*/*CDF* distributions between the sample and the population. Here, the nodes' degree, shortest-path, clustering-coefficient and k-core distributions are considered. Note that both H and K distributions in our cases are discrete distributions, not continuous distributions, so there is no parameter that needs to be given beforehand.

6.3.2. Quality measures of clustering structure

The basic properties mentioned above and the corresponding measures of representativeness are inadequate for the essential property: clustering structure. Therefore, our evaluation methodology on how to evaluate the clustering structure quantitatively is described in this subsection.

Methodology. Firstly, clusters of each snapshot in the original graph stream are produced to serve as *ground-truth* in which any credible clustering algorithm can be employed. Note that to obtain method-independent results, two credible clustering algorithms (i.e., *Blondel* [6] and *BigClam* [29]) are employed to produce the ground-truth clusters. Secondly, the same clustering algorithm is executed on the subgraph yielded by each sampling technique. In this way, the clustering structure is obtained in the sampled subgraph. Thirdly, the clustering quality of the subgraph is evaluated by multiple metrics such that we can validate the effectiveness of them.

Measurements of clustering. Briefly, we describe several clustering quality metrics to evaluate how representative the sampled counterpart is with regard to the clustering structure. Since *precision* and *recall* are considered to be two important aspects of clustering quality, each aspect should be dealt with individually without losing both's importance. For this reason, foremost, we adopt δ-*precison* and δ-*recall* [31,34] to evaluate the sampling quality between the population's clustering structure and that of the sampled subgraph. δ is a predefined purity threshold and it controls the matching degree between clusters of sampled counterpart and those of original graph. If the matching degree is not less than δ, two clusters are identified as a right match. Higher value of δ-*recall* implies the ground-truth clusters of G are more successfully covered by the obtained clusters of G_s while higher value of δ-*precision* indicates that the obtained clusters of G_s are more precisely representative of the ground-truth clusters of G [34].

Second, multiple classic metrics, i.e., *NMI* [10], *ARI* [10], and *accuracy for number of clusters (ANC)* [29], are extensively adopted for the graph clustering evaluation. Thus they are also utilized to evaluate the clustering results of the sampled graphs. However, these metrics are only devised to estimate the clustering quality on the whole graph, not particularly the sampled counterpart. For comparison purposes, we exploit them on the subgraph $G(V_S)$ (i.e., the same node-set in the sample).

6.4. Experimental results

6.4.1. Overall performance

In the first experiment, we sample 20% percent of total number of nodes appearing in the stream. To evaluate the quality of the sample, Multiple snapshots are taken at different time-stamps according to their stream lengths. Various quality metrics are calculated for each designated snapshot and the mean values are taken for all the snapshots to evaluate the overall performance. We record these quality metrics and the sampling time of each algorithm for synthetic graphs in Table 3 and real-world graphs in Table 4. Here experimental results using *Blondel* clustering are presented and other results using *BigClam* algorithm [29] exhibit similar behaviors. The experimental results are as follows:

- Generally, for the clustering metrics (i.e., δ-*precison,* δ-*recall, NMI* and *ARS*), *CPIES* outperforms other sampling algorithms in most cases. Unlike *PIES* method, it is also good at preserving the inherent clustering structure in the dense, tightly-knit graphs (e.g., *Enron-employee* and *Email-Eu-core*). The reason is that *CPIES* is biased towards the hub nodes with high degrees, and they are good representatives of the underlying clustering structure. For *PIES* algorithm, the random replacement strategy causes that the replaceable nodes may be the hub nodes of clusters. It could damage the clustering structure and make the sample less of important nodes and their important links. Since *StreamNS* and *StreamES* sample the nodes/edges uniformly regardless of network's inherent structure, their sample qualities depend on the structures of graphs to a large extent. Besides, we derived that sampling methods which contain the induced edge step (i.e., *CPIES, PIES, PIES (Min)* and *StreamNS*) exhibit better results than *StreamES* (under-sample of edges) in most cases since they accommodate more edges that are associated to the sampled nodes.
- In terms of KS-divergence and JS-divergence of selected distributions, generally for degree distribution, we observe that: (i) *StreamNS* performs slightly better than *CPIES* and *PIES*. *StreamES* heavily underestimates the degree distribution and performs the worst; (ii) for clustering-coefficient and k-core distributions, *CPIES* outperforms other samplings and *PIES* comes in the second rank. *StreamNS* is apt to capture the distribution of degree but underestimates the distribution of k-core distributions and clustering-coefficient; (iii) for the shortest-path distribution, they do not make much differences except for *StreamES* which produces samples containing multiple small connected components.
- It is interesting to observe that most distribution divergences for all samplings deteriorate when the graph becomes tightly-knit clustered and more dense. It means that the tightly-knit clustered graphs are more difficult to preserve the topological structure than the sparse ones. From the clustering metrics (e.g., *NMI*), we can see that *CPIES* performs better than other samplings on those graphs which means *CPIES* is also suitable to sample dense and tightly-clustered streaming graph.
- We can observe that the *CPIES* algorithm performs almost as well as the *PIES* algorithm in terms of the execution time. The *StreamES* method is the fastest algorithm while *StreamNS* ranks the second. It is because that *StreamES* does not invoke an induced edge step and it costs less time than other algorithms. However, *StreamES* does not induce extra edges through partial edge induction such that it greatly underestimates the number of edges in the graphs.

Table 3
The qualities of various sampling strategies on synthetic graphs ($p = 20\%$). All the clusters are produced by *Blondel* clustering on both original graph and the sampled counterpart and bold values indicate the best results for corresponding metrics. Please note that for divergence of distributions, smaller value indicates better fitness between two distributions while for clustering qualities, larger value means that the clustering structure is preserved better. Moreover, the maximum value of δ-*recall* is equal to the sample rate p.

(a) MS	Measurements	CPIES	PIES	StreamNS	StreamES	PIES(Min)	(b) EC	Measurements	CPIES	PIES	StreamNS	StreamES	PIES(Min)
Distribution	JS_degree	0.962	0.961	0.984	1.000	**0.956**	Distribution	JS_degree	0.966	0.966	0.966	1.000	**0.965**
	KS_degree	0.985	0.985	0.994	1.000	**0.972**		KS_degree	0.987	0.987	**0.986**	1.000	0.987
	JS_CC	**0.628**	0.638	0.655	0.671	0.648		JS_CC	**0.701**	0.707	0.708	0.735	0.707
	KS_CC	**0.673**	0.734	0.727	0.843	0.722		KS_CC	**0.723**	0.759	0.741	0.886	0.750
	JS_path	0.558	0.533	0.621	0.593	**0.520**		JS_path	0.569	**0.553**	0.613	0.581	0.572
	KS_path	0.718	**0.666**	0.773	0.740	0.682		KS_path	0.718	0.693	0.767	**0.629**	0.719
	JS_kcores	**0.471**	0.478	0.486	0.655	0.471		JS_kcores	**0.469**	0.487	0.471	0.641	0.489
	KS_kcores	**0.669**	0.678	0.681	0.833	0.670		KS_kcores	0.673	0.685	**0.667**	0.826	0.681
Clustering metrics	1.0-precision	**0.532**	0.517	0.388	0.457	0.432	Clustering metrics	1.0-precision	**0.505**	0.462	0.426	0.458	0.459
	1.0-recall	**0.041**	0.039	0.035	0.040	0.032		1.0-recall	**0.047**	0.031	0.034	0.031	0.029
	0.5-precision	**0.762**	0.728	0.686	0.699	0.718		0.5-precision	0.708	0.665	0.701	**0.719**	0.702
	0.5-recall	**0.088**	0.088	0.084	0.050	0.082		0.5-recall	**0.097**	0.087	0.092	0.048	0.092
	0.0-precision	**0.794**	0.764	0.734	0.699	0.783		0.0-precision	0.744	0.712	**0.750**	0.719	0.723
	0.0-recall	**0.120**	0.117	0.117	0.050	0.118		0.0-recall	**0.131**	0.120	0.115	0.048	0.120
	ANC	0.622	0.576	**0.836**	0.211	0.792		ANC	**0.784**	0.645	0.635	0.212	0.659
	NMI	**0.742**	0.698	0.697	0.683	0.723		NMI	**0.779**	0.707	0.740	0.742	0.708
	ARS	**0.391**	0.341	0.376	0.129	0.322		ARS	**0.483**	0.390	0.470	0.139	0.393
Sample statistics	S_#edges	209	198	176	110	205	Sample statistics	S_#edges	216	209	181	112	220
	S_#nodes	200	200	170	200	200		S_#nodes	200	200	166	200	200
	S_modularity	0.736	0.732	0.775	0.165	0.702		S_modularity	0.759	0.750	0.791	0.229	0.752
	S_density	0.033	0.034	0.021	0.440	0.032		S_density	0.024	0.028	0.018	0.384	0.0252
	S_CC	0.070	0.043	0.049	0.000	0.056		S_CC	0.054	0.052	0.064	0.000	0.053
	S_#clusters	34	36	23	90	35		S_#clusters	35	39	25	87	34
	Sample time	0.38	0.39	0.09	0.13	2.42		Sample time	0.42	0.41	0.07	0.11	2.63

Table 4
The qualities of various sampling strategies on real-world graphs ($p = 20\%$). All the clusters are produced by *Blondel* clustering on both original graph and the sampled counterpart and bold values indicate the best results for corresponding metrics. Please note that for divergence of distributions, smaller value indicates better fitness between two distributions while for clustering qualities, larger value means that the clustering structure is preserved better. Moreover, the maximum value of δ-*recall* is equal to the sample rate p.

(a) Enron-employee	Measurements	CPIES	PIES	StreamNS	StreamES	PIES(Min)	(b) Email-Eu-email	Measurements	CPIES	PIES	StreamNS	StreamES	PIES(Min)
Distributions	JS_degree	0.421	0.428	**0.416**	0.583	0.436	Distributions	JS_degree	**0.289**	0.357	0.366	0.639	0.330
	KS_degree	0.512	0.523	**0.509**	0.688	0.517		KS_degree	**0.463**	0.553	0.561	0.811	0.516
	JS_CC	**0.424**	0.457	0.446	0.601	0.487		JS_CC	**0.465**	0.488	0.485	0.718	0.475
	KS_CC	**0.301**	0.337	0.316	0.698	0.386		KS_CC	**0.185**	0.357	0.339	0.875	0.308
	JS_path	0.141	0.102	**0.065**	0.206	0.127		JS_path	**0.044**	0.235	0.250	0.143	0.171
	KS_path	0.261	0.204	**0.179**	0.371	0.232		KS_path	**0.164**	0.468	0.484	0.233	0.385
	JS_kcores	**0.240**	0.244	0.309	0.504	0.251		JS_kcores	**0.410**	0.497	0.507	0.776	0.459
	KS_kcores	**0.374**	0.379	0.458	0.647	0.374		KS_kcores	**0.603**	0.690	0.699	0.904	0.651
Clustering metrics	1.0-precision	0.683	0.662	0.547	**0.704**	0.684	Clustering metrics	1.0-precision	0.242	0.347	0.326	**0.589**	0.383
	1.0-recall	0.114	0.110	0.085	0.103	**0.118**		1.0-recall	**0.041**	0.025	0.021	0.028	0.027
	0.5-precision	**0.868**	0.840	0.807	0.764	0.862		0.5-precision	0.729	**0.736**	0.716	0.690	0.729
	0.5-recall	**0.155**	0.144	0.115	0.106	0.150		0.5-recall	**0.117**	0.098	0.098	0.031	0.099
	0.0-precision	**0.883**	0.852	0.832	0.764	0.879		0.0-precision	**0.785**	0.774	0.770	0.708	0.773
	0.0-recall	**0.164**	0.151	0.139	0.106	0.154		0.0-recall	**0.131**	0.116	0.117	0.031	0.111
	ANC	**0.861**	0.857	0.821	0.548	0.825		ANC	**0.845**	0.549	0.543	0.106	0.540
	NMI	**0.784**	0.766	0.751	0.736	0.770		NMI	**0.611**	0.546	0.555	0.504	0.545
	ARS	**0.646**	0.606	0.561	0.343	0.595		ARS	**0.439**	0.364	0.362	0.043	0.365
Sample statistics	S_#edges	40	38	32	19	37	Sample statistics	S_#edges	626	477	467	128	529
	S_#nodes	28	28	23	28	28		S_#nodes	170	196	197	197	197
	S_modularity	0.404	0.396	0.382	0.205	0.323		S_modularity	0.514	0.617	0.633	0.389	0.571
	S_density	0.216	0.186	0.172	0.422	0.219		S_density	0.044	0.027	0.026	0.209	0.030
	S_CC	0.355	0.329	0.280	0.038	0.271		S_CC	0.346	0.287	0.293	0.000	0.310
	S_#clusters	6	6	5	12	7		S_#clusters	9	14	14	72	15
	Sample time	1.19	0.09	0.32	0.01	10.18		Sample time	23.51	22.13	5.49	0.3	192.47
(c) ColleageMsg	Measurements	CPIES	PIES	StreamNS	StreamES	PIES(Min)	(d) Facebook-wall	Measurements	CPIES	PIES	StreamNS	StreamES	PIES(Min)
Distributions	JS_degree	0.163	0.125	0.135	0.363	**0.092**	Distributions	JS_degree	**0.010**	0.021	0.014	0.072	0.021
	KS_degree	0.324	0.255	0.257	0.573	**0.185**		KS_degree	**0.050**	0.078	0.058	0.153	0.077
	JS_CC	0.248	0.203	0.207	0.397	**0.174**		JS_CC	**0.013**	0.027	0.017	0.108	0.027
	KS_CC	0.414	0.323	**0.278**	0.600	0.215		KS_CC	**0.052**	0.084	0.057	0.198	0.083
	JS_path	0.175	0.079	0.052	0.267	**0.034**		JS_path	**0.017**	0.050	0.027	0.258	0.049
	KS_path	0.390	0.267	0.217	0.345	**0.143**		KS_path	**0.064**	0.133	0.094	0.323	0.130
	JS_kcores	0.404	0.348	0.335	0.650	**0.261**		JS_kcores	**0.020**	0.059	0.042	0.138	0.058
	KS_kcores	0.596	0.538	0.516	0.826	**0.429**		KS_kcores	**0.059**	0.125	0.099	0.219	0.123
Clustering metrics	1.0-precision	**0.397**	0.149	0.120	0.307	0.192	Clustering metrics	1.0-precision	**0.961**	0.940	0.955	0.903	0.941
	1.0-recall	**0.054**	0.002	0.002	0.012	0.011		1.0-recall	**0.170**	0.158	0.169	0.145	0.158
	0.5-precision	0.455	0.222	0.210	**0.478**	0.281		0.5-precision	**0.975**	0.958	0.973	0.935	0.959
	0.5-recall	**0.060**	0.008	0.009	0.018	0.019		0.5-recall	**0.172**	0.159	0.171	0.146	0.160
	0.0-precision	**0.571**	0.417	0.452	0.515	0.480		0.0-precision	**0.980**	0.963	0.978	0.940	0.963
	0.0-recall	**0.093**	0.046	0.048	0.023	0.054		0.0-recall	**0.172**	0.160	0.172	0.146	0.161
	ANC	0.437	0.789	**0.913**	0.128	0.809		ANC	**0.951**	0.928	0.856	0.795	0.931

(*continued on next page*)

Table 4 (*continued*)

(a) Enron-employee	Measurements	CPIES	PIES	StreamNS	StreamES	PIES(Min)	(b) Email-Eu-email	Measurements	CPIES	PIES	StreamNS	StreamES	PIES(Min)
	NMI	0.235	0.172	0.205	**0.391**	0.171		NMI	**0.781**	0.759	0.779	0.742	0.763
	ARS	0.043	0.043	**0.055**	0.017	0.049		ARS	**0.699**	0.666	0.686	0.597	0.669
Sample statistics	S_#edges	690	918	691	283	1217	Sample statistics	S_#edges	13,383	10,842	11,001	7610	10,867
	S_#nodes	379	379	295	379	379		S_#nodes	5635	5635	5146	56,357	5635
	S_modularity	0.515	0.441	0.435	0.545	0.360		S_modularity	0.616	0.631	0.623	0.690	0.632
	S_density	0.012	0.013	0.016	0.111	0.018		S_density	0.095	0.095	0.095	0.095	0.095
	S_CC	0.040	0.048	0.076	0.000	0.066		S_CC	0.078	0.070	0.075	0.038	0.070
	S_#clusters	31	17	14	104	16		S_#clusters	364	352	281	667	356
	Sample time	3.62	3.58	1.02	0.24	28.32		Sample time	399.14	396.37	10.48	2.77	480.83
(e) Slashdot	Measurements	CPIES	PIES	StreamNS	StreamES	PIES(Min)	(f) Reality	Measurements	CPIES	PIES	StreamNS	StreamES	PIES(Min)
Distributions	JS_degree	**0.005**	0.005	0.009	0.021	0.008	Distributions	JS_degree	0.022	0.022	**0.020**	0.025	0.023
	KS_degree	0.020	**0.019**	0.054	0.102	0.057		KS_degree	0.030	0.031	**0.017**	0.029	0.041
	JS_CC	0.004	**0.003**	0.009	0.043	0.009		JS_CC	**0.010**	0.010	0.008	0.011	0.013
	KS_CC	0.023	**0.020**	0.044	0.097	0.043		KS_CC	0.012	0.012	**0.008**	0.022	0.029
	JS_path	0.014	0.016	0.026	0.449	**0.009**		JS_path	0.049	0.047	0.086	0.359	**0.008**
	KS_path	0.093	0.099	0.124	0.612	**0.075**		KS_path	0.163	0.157	0.203	0.514	**0.040**
	JS_kcores	0.043	0.039	0.064	0.161	**0.005**		JS_kcores	**0.005**	0.006	0.012	0.021	0.005
	KS_kcores	0.109	0.099	0.159	0.305	**0.049**		KS_kcores	**0.015**	0.017	0.039	0.059	0.050
Clustering metrics	1.0-precision	**0.877**	0.867	0.851	0.834	0.811	Clustering metrics	1.0-precision	0.514	0.498	**0.549**	0.485	0.471
	1.0-recall	**0.022**	0.021	0.021	0.020	0.020		1.0-recall	**0.083**	0.077	0.073	0.075	0.080
	0.5-precision	**0.892**	0.882	0.866	0.875	0.842		0.5-precision	**0.925**	0.835	0.832	0.852	0.797
	0.5-recall	0.027	0.027	0.028	0.024	**0.030**		0.5-recall	**0.147**	0.144	0.134	0.140	0.140
	0.0-precision	**0.911**	0.903	0.895	0.888	0.873		0.0-precision	0.861	0.868	**0.892**	0.880	0.842
	0.0-recall	0.058	0.059	0.062	0.041	**0.066**		0.0-recall	**0.184**	0.180	0.140	0.164	0.184
	ANC	0.205	0.220	0.244	0.140	**0.322**		ANC	**0.886**	0.867	0.652	0.869	0.792
	NMI	**0.421**	0.362	0.419	0.419	0.345		NMI	0.885	0.879	**0.892**	0.860	0.876
	ARS	**0.158**	0.147	0.154	0.120	0.157		ARS	0.728	0.714	**0.748**	0.620	0.705
Sample statistics	S_#edges	15,745	16,392	11,558	10,588	21,858	Sample statistics	S_#edges	1495	1492	989	1383	1581
	S_#nodes	9876	9875	8057	9876	9876		S_#nodes	1359	1357	938	1361	1361
	S_modularity	0.623	0.615	0.686	0.827	0.511		S_modularity	0.864	0.863	0.871	0.909	0.834
	S_density	0.00039	0.00039	0.00042	0.00034	0.00049		S_density	0.002	0.002	0.003	0.002	0.002
	S_CC	0.017	0.016	0.014	0.004	0.022		S_CC	0.019	0.018	0.017	0.005	0.027
	S_#clusters	536	420	340	1095	274		S_#clusters	38	38	27	49	33
	Sample time	326.44	251.55	3.96	3.11	1500.56		Sample time	8.82	9.58	1.02	0.15	63.32

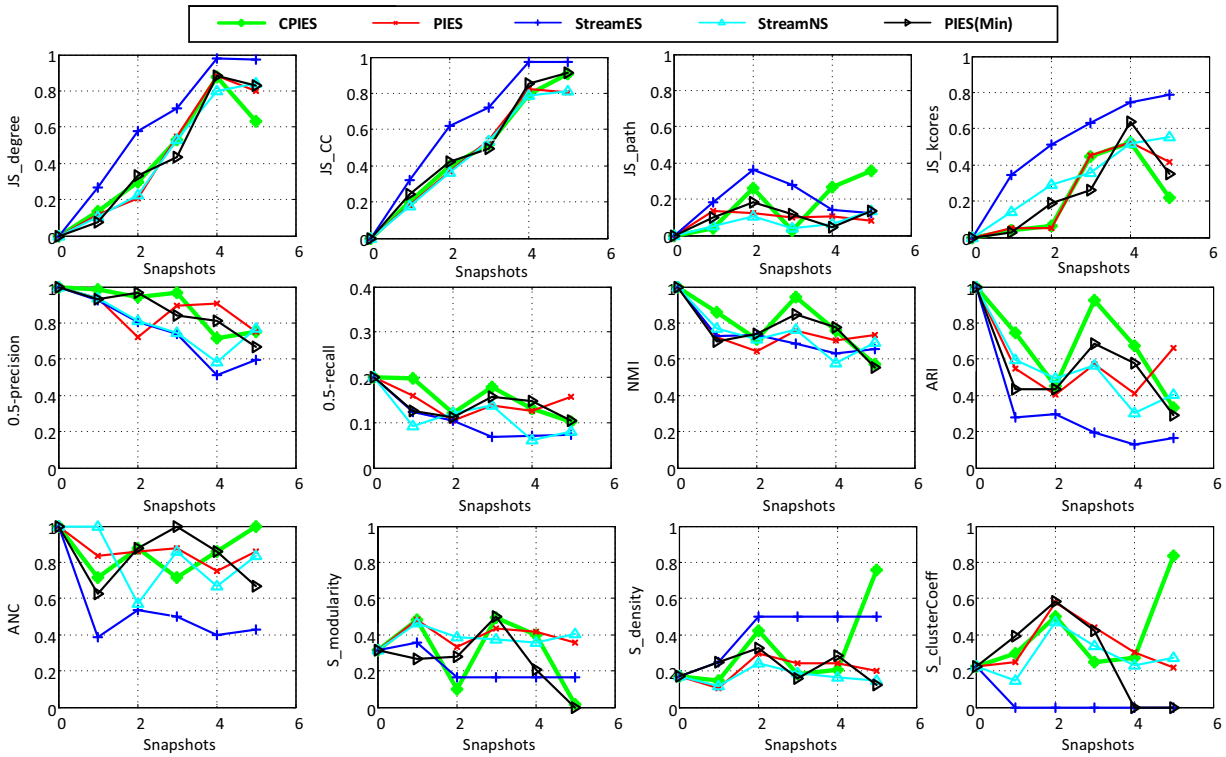

Fig. 1. Quality comparison of different sampling algorithms on the *Enron-employee* network in multiple snapshots ($p = 0.2$).

Note that isolated nodes analysis is carried out for both *CPIES* and *PIES*. Since the basic *PIES* requires to substitute nodes randomly, it is not surprise that its sampled subgraph incorporates isolated nodes (i.e., their degree is equal to zero). Once the new edge is sampled, randomly selected nodes are replaced by the new edge's incident nodes. This random strategy could substitute nodes with high degree whereas isolated nodes are kept. The proposed *CPIES*, however, avoids isolated nodes completely and makes the sampling quality significantly improved.

6.4.2. Evaluation on designated snapshots of graphs

In the second experiment, we further show the performances of various sampling algorithms on designated snapshots of streaming graphs, and conduct the experiments on both *Enron-employee* and *Email-Eu-core* networks. Figs. 1 and 2 show the *JS-divergences* of selected distributions and clustering quality metrics at designated snapshots in the streams. We can draw the following conclusions:

- The figures follow the same trends as the effects on most of real-world graphs. *CPIES* performs better than *PIES, StreamNS* and *StreamES* in most of snapshots on both real graphs. It illustrates that *CPIES* sampling can consistently maintain clustering representative samples at different snapshot moments.
- The results demonstrate that *CPIES* performs well when the graph is dense and tightly-clustered. With the arrival of new edges, it always maintains the hub nodes and discards the peripheral nodes of clusters in the stream. Thus it is capable of preserving the clustering structure well and capturing the evolution of clusters.

6.4.3. Impacts of sampling rates

In the third experiment, the impact of the sample rate p is carried out and the sample rate p ranges between 0.20 and 0.80 with the interval of 0.20. 5 trials are run for each sampling algorithm and the mean value of each metric are calculated. The results on *Enron-employee, Reality* and *ColleageMsg* are shown in Figs. 3–5, respectively.

We mainly can observe that: (i) performances of all the sampling algorithms are improved as the sample rate increases; (ii) for clustering quality metrics, *CPIES* is capable of consistently obtaining good sampling qualities with different sampling rates. Also, *StreamES* does not invoke an induced edge step, so the number of edges is heavily underestimated and it cannot retain the clustering structure well.

6.4.4. Distributions for final streams (at 20% sample rate)

In the fourth experiment, we plot the complementary cumulative distribution functions (*CCDF*) of four distributions (i.e., degree, shortest-path, clustering-coefficient and k-core distribution) for these real-world graphs in Figs. 6–10. The sample

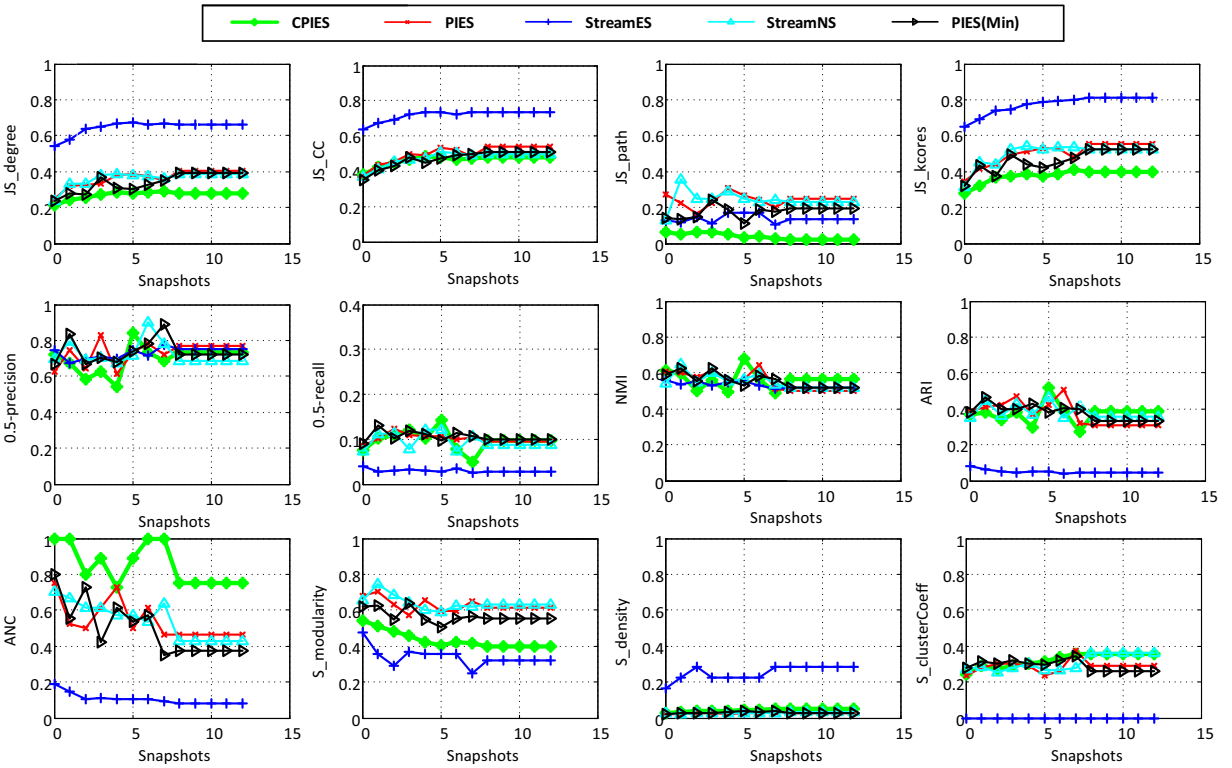

Fig. 2. Quality comparison of different sampling algorithms on the *email-EU-core* network in multiple snapshots ($p = 0.2$).

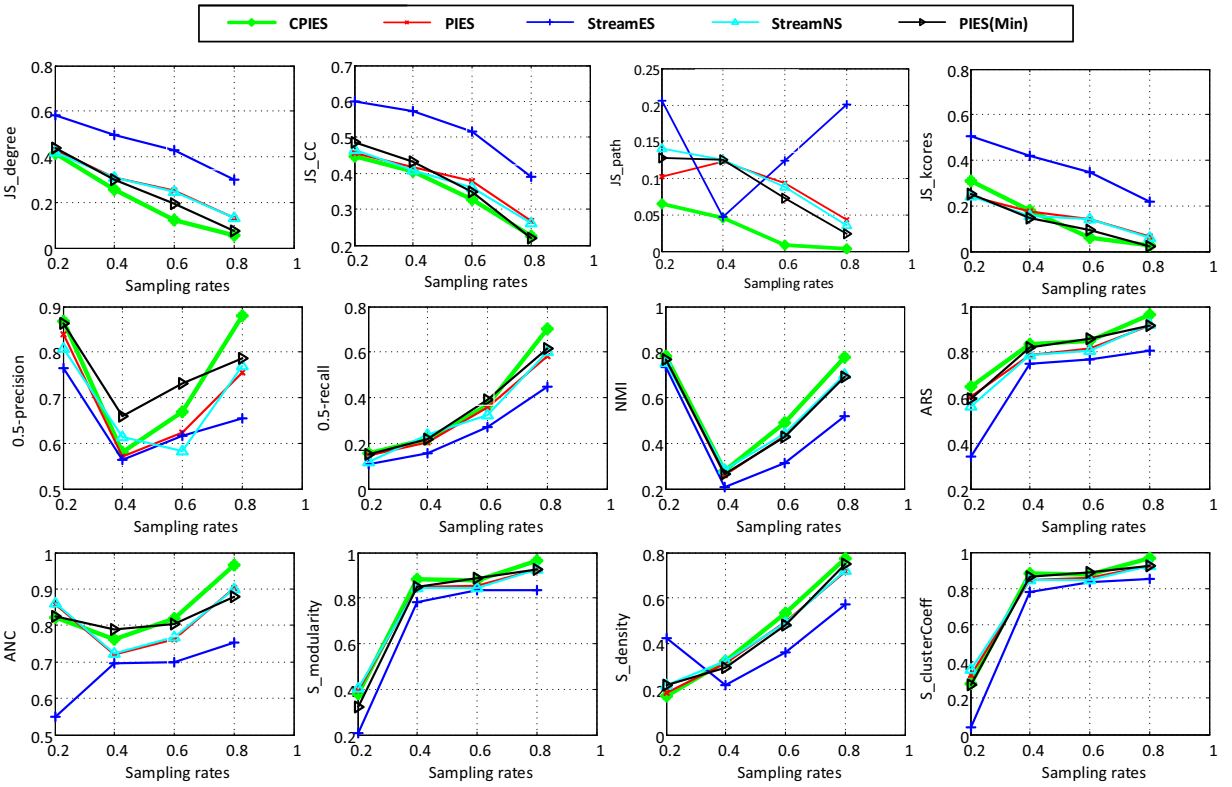

Fig. 3. The impact of varying sample rate p on the quality metrics on the *Enron-employee* networks with *Blondel* clustering.

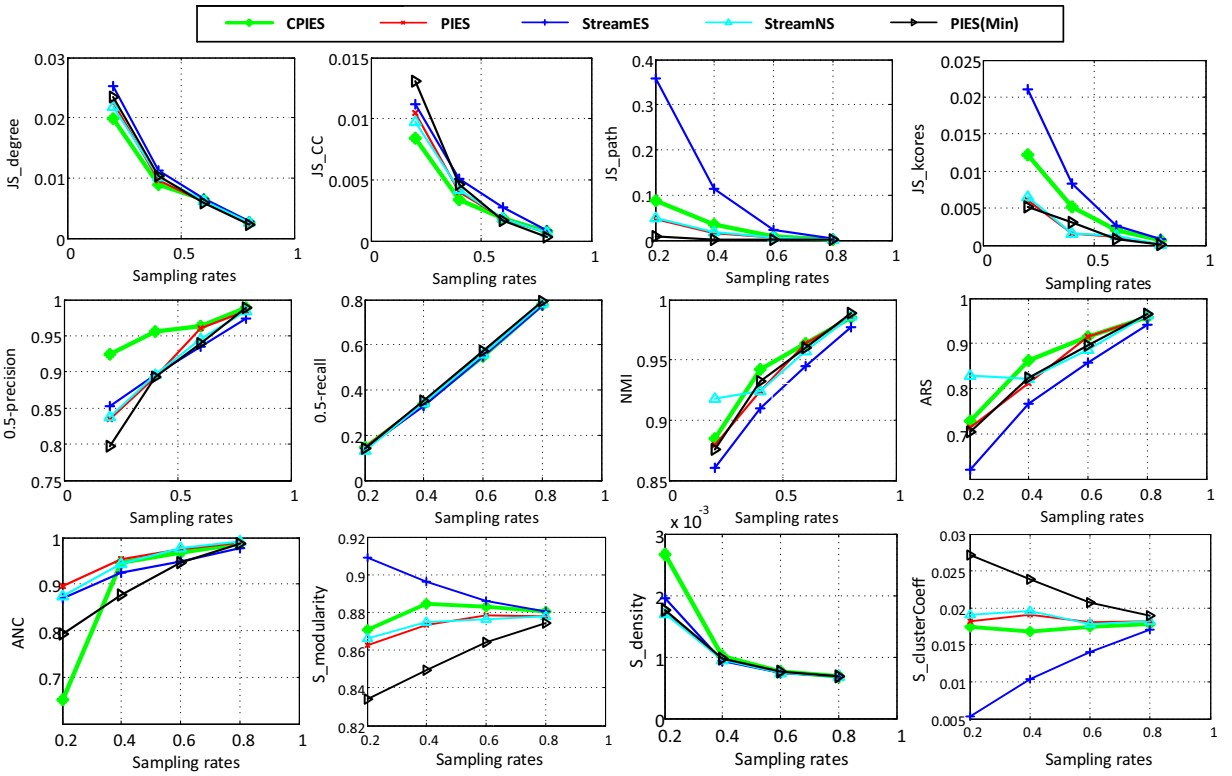

Fig. 4. The impact of varying sample rate p on the quality metrics on the *Reality* networks with *Blondel* clustering.

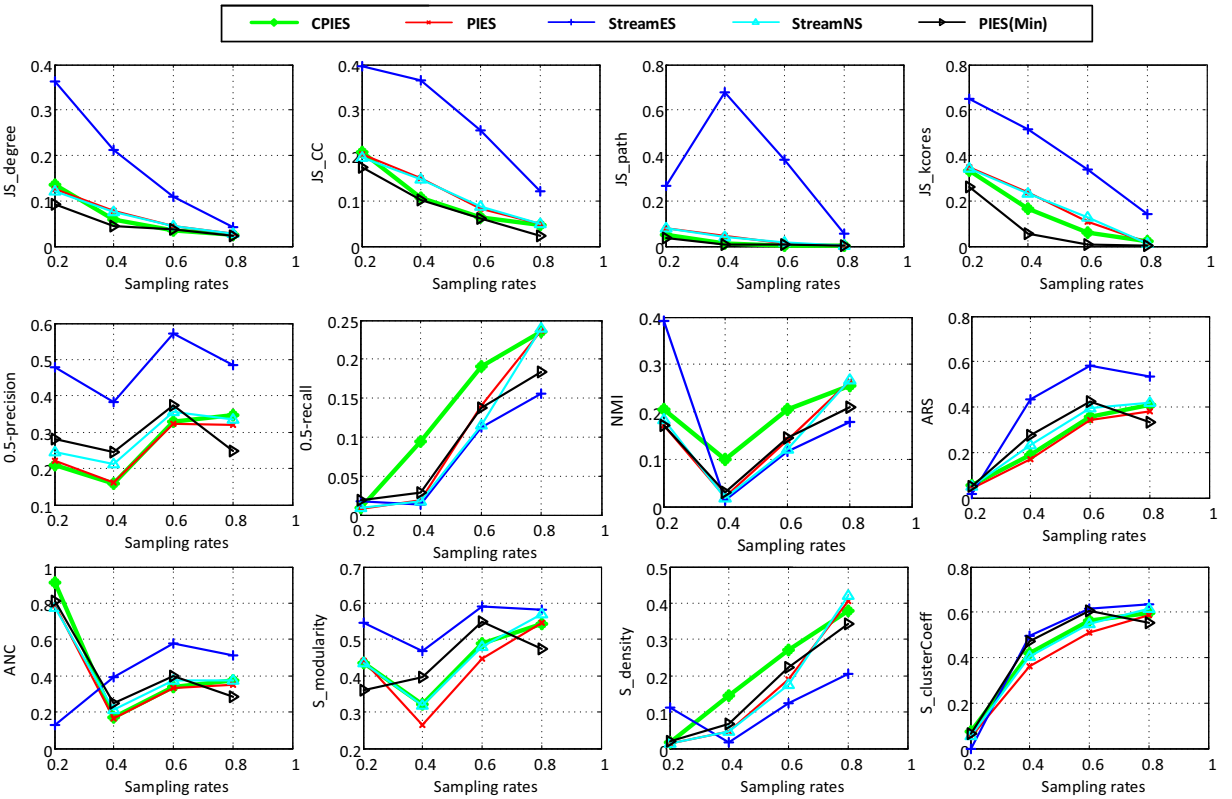

Fig. 5. The impact of varying sample rate p on the quality metrics on the *ColleageMsg* network with *Blondel* clustering.

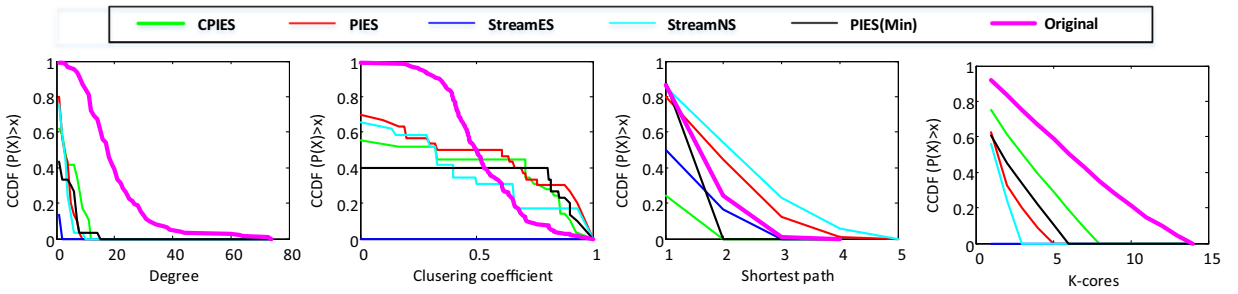

Fig. 6. Distribution function at 20% sampling rate on the *Enron-employee* networks.

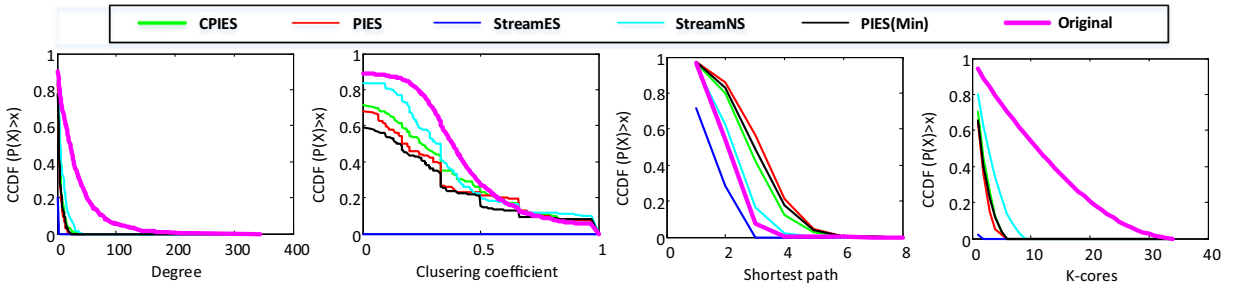

Fig. 7. Distribution function at 20% sampling rate on the *Email-Eu-core* networks.

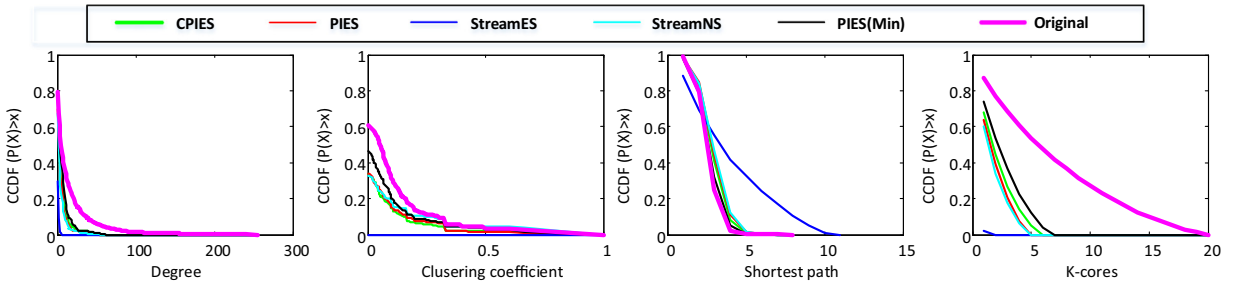

Fig. 8. Distribution function at 20% sampling rate on the *ColleageMsg* networks.

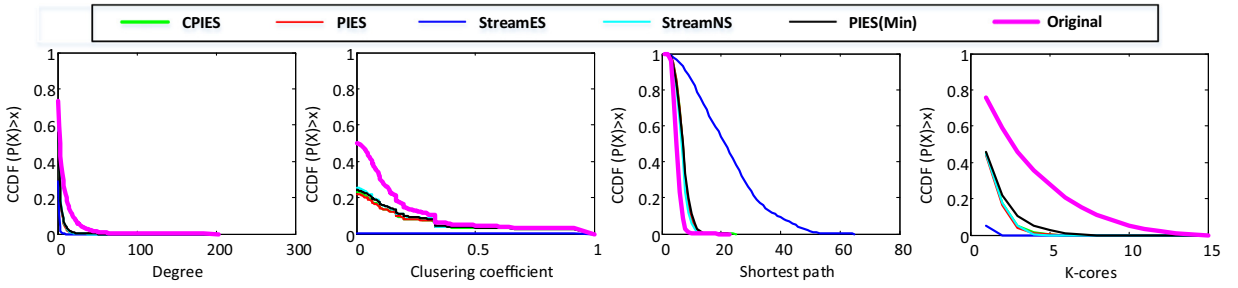

Fig. 9. Distribution function at 20% sampling rate on the *Facebook* networks.

rate here is 20% and other sampling rates show similar behaviors. All the distributions are constructed by using the end stream (i.e., 100% length of the stream). Here principal findings from these distributions are summed up.

Degree distribution: In general, we can observe that all the sampling algorithms under-estimate the degree of the nodes. However, we note that the proposed *CPIES* algorithm is more biased towards high degree nodes of different clusters and eliminates nodes with low degrees. *StreamNS* is not biased to wards high degree nodes and its distribution is well matched and has the similar trend (slope) as the original degree distribution. *StreamES* algorithm performs the worst and causes a large amount of low-degree nodes in its sample, thus the slope of degree distribution is a sharp decline.

Clustering-coefficient distribution: We can observe that *CPIES* performs better than *PIES* which is lack of the hub nodes of clustering structures. This behavior is more distinguishable in more dense and clustered network such as *Email-Eu-core*

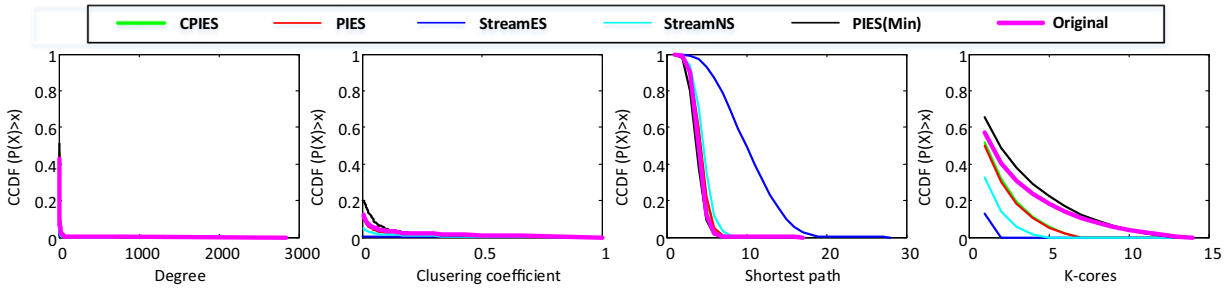

Fig. 10. Distribution function at 20% sampling rate on the *Slash* networks.

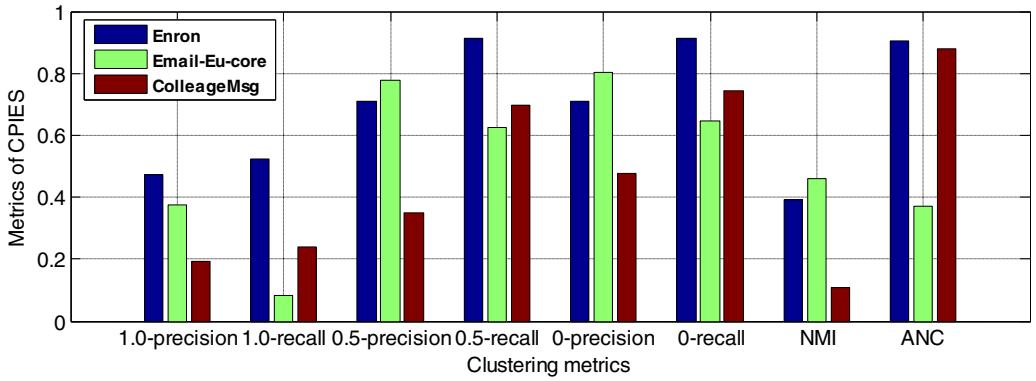

Fig. 11. Inferring clustering affiliation for real-world graphs using *CPIES* method at 20% sampling rate.

networks. *StreamNS* performs better than *StreamES* which produces unclustered samples. *StreamES* gives too sparse samples which contain no local clustering-coefficient under small sampling size.

Shortest path distribution*: CPIES* is able to preserve the shortest-path distribution as good as *PIES* algorithms for most graphs. It can be observed that most of the sampling algorithms overestimate the shortest-path except for *StreamES*. It is because *StreamES* under-samples the edges and have multiple connected components. The shortest-path is only calculated in the maximum component such that *StreamES* has shorter paths than other sampling algorithms.

K-core distribution*: CPIES* is superior to the competing methods for tightly-clustered graphs (e.g., *Enron-employee*). For less-clustered graphs, *CPIES* performs almost as good as *PIES (Min)*. It is because *CPIES* is bias toward high-degree nodes which results in maintaining the k-core structure well.

Overall, *CPIES* is the closest to preserving the cluster-related distributions (i.e., clustering-coefficient and k-core) compared to other sampling methods. This is due to the fact that *CPIES* samples representative nodes with a larger probability than *StreamNS* and *PIES*. *StreamES* does not induce extra edges through partial edge induction such that it greatly underestimates the number of edges in the graphs.

6.4.5. Inferring clustering affiliations for the population

In previous sections, we evaluated how representative the sample is according to the original population. Here we utilize *CPIES* algorithm to produce samples at the sample rate 20% and then employ the two-stage inference method to infer clustering affiliation for the un-sampled nodes of the population.

The clustering inference results from 20% samples of population graphs are shown in Fig. 11. We can derive that the proposed *CPIES* approach combined with the two-stage inference performs well on inferring the clustering affiliation for real-world graphs. These results further validate that *CPIES* sampling produces samples which are representatives of inherent clustering structure and the two-stage inference method provides a feasible solution to infer clustering labels of the population graphs. The described results are of value to the community as a basis for understanding the merits of the approach.

6.5. Summary of experimental evaluation

To evaluate our proposed algorithm's performance, five groups of extensive experiments were conducted on both synthetic and real-world graphs. The experiments were conducted to evaluate the sample's representativeness of basic properties and the inherent clustering structure, respectively. We can briefly summarize remarkable findings in each experiment.

In the first experiment, we described the overall performances of the proposed *CPIES* algorithm and competing algorithms. From these experiments, we observed that *CPIES* obtained better results in preserving the clustering structure compared with other algorithms. The reason is that *CPIES* is inclined to sample nodes with high-degree and exclude the isolated nodes, thus it produces sampled subgraphs which preserve topological properties more accurately.

In the second experiment, we investigated how the sample qualities of subgraphs produced by various sampling methods fluctuate on designated snapshots of streaming graphs. The results indicated that *CPIES* algorithm can consistently maintain clustering representative samples at different lengths of the stream.

In the third experiment, we analyzed the impact of the sample rate on the quality of the sample. Performances of all the sampling algorithms are improved as the sample rate increases. Moreover, we also observed that *CPIES* algorithm can obtain good clustering representative samples just at the sampling rate of 20%.

In the fourth experiment, we drew complementary cumulative distribution functions of four basic distributions. We found that *CPIES* is the closest to preserve two cluster-related distributions (i.e., clustering-coefficient and k-core) in most cases. This is due to the fact that *CPIES* method samples representative nodes in each cluster with a larger probability compared to *PIES*. In contrast, *PIES* contains lower percentage of high degree nodes, and has a large number of isolated nodes and disjoint connected components in the sample.

Finally, in the last experiment, we employed the two-stage inference method to infer clustering affiliations for the un-sampled nodes of the population. These results showed that the two-stage clustering inference method in conjunction with *CPIES* sampling provides a feasible solution to infer clustering labels of the population.

7. Conclusion

In this paper, a novel cluster-preserving sampling *PIES* algorithm was proposed to keep an up-to-date sample while preserving the clustering structure of the fully-dynamic streaming graph. It is capable of retaining clusters' hub nodes and eliminating isolated nodes, which makes the population graph's clustering structure preserved in the sample. The isolated nodes elimination mechanism is simple yet efficient and compatible well with most node replacement strategies, which makes it more versatile and feasible. Also, *CPIES* can handle the edge-deletion requests which is of crucial importance in a fully-dynamic streaming setting. Furthermore, we also propose a new two-stage clustering inference approach to infer the un-sampled nodes in the population.

The experiments on various benchmarks and real-world graphs demonstrated that the proposed *CPIES* algorithm is capable of preserving the inherent clustering structure while sampling from the fully-dynamic streaming graphs, and performs better than other competing algorithms in cluster-related properties. Moreover, the proposed *CPIES* combined with *TCI* method provides a feasible solution to inference clustering affiliations of the population.

Our future work is to investigate more evaluation metrics to quantitatively evaluate the sample's quality. Such evaluation metrics will guide our further understanding of enhanced sampling approaches.

Acknowledgments

This research of Zhang is supported by the Foundation for Innovative Research Groups of the National Natural Science Foundation of China (Grant no. 61521003) and the National Key Research and Development Program of China (Grant no. 2016YFB0800101).

References

[1] C.C. Aggarwal, Y. Zhao, P.S. Yu, Outlier detection in graph streams, in: ICDE, IEEE, 2011, pp. 399–409.
[2] N.K. Ahmed, F. Berchmans, J. Neville, R. Kompella, Time-based sampling of social network activity graphs, in: Proceedings of the Eighth Workshop on Mining and Learning with Graphs, ACM, 2010, pp. 1–9.
[3] N.K. Ahmed, N. Duffield, J. Neville, R. Kompella, Graph sample and hold: a framework for big-graph analytics, in: KDD, ACM, 2014, pp. 1446–1455.
[4] N.K. Ahmed, N. Duffield, T.L. Willke, R.A. Rossi, On sampling from massive graph streams, in: Proceedings of the VLDB Endowment, 10(11), 2017, pp. 1430–1441.
[5] N.K. Ahmed, J. Neville, R. Kompella, Network sampling: from static to streaming graphs, ACM Trans. Knowl. Discov. Data 8 (2) (2014) 7.
[6] V.D. Blondel, J.-L. Guillaume, R. Lambiotte, E. Lefebvre, Fast unfolding of communities in large networks, J. Stat. Mech. 10 (2008) 008.
[7] E. Cem, M.E. Tozal, K. Sarac, Impact of sampling design in estimation of graph characteristics, in: IPCCC, IEEE, 2013, pp. 1–10.
[8] S. Chib, E. Greenberg, Understanding the metropolis-hastings algorithm, Am. Stat. 49 (4) (1995) 327–335.
[9] P. Ebbes, Z. Huang, A. Rangaswamy, Subgraph sampling methods for social networks: the good, the bad, and the ugly, HEC Paris Research Paper No. MKG-2014-1027, 2010.
[10] S. Fortunato, D. Hric, Community detection in networks: a user guide, Phys. Rep. 659 (2016) 1–44.
[11] M. Gjoka, M. Kurant, C.T. Butts, A. Markopoulou, Walking in facebook: a case study of unbiased sampling of osns, in: Infocom, IEEE, 2010, pp. 1–9.
[12] D. Greene, D. Doyle, P. Cunningham, Tracking the evolution of communities in dynamic social networks, in: ASONAM, IEEE, 2010, pp. 176–183.
[13] P. Hu, W.C. Lau, A survey and taxonomy of graph sampling, arXiv:1308.5865 (2013).
[14] E.M. Jin, M. Girvan, M.E. Newman, Structure of growing social networks, Phys. Rev. E 64 (4) (2001) 046132.
[15] J. Leskovec, C. Faloutsos, Sampling from large graphs, in: KDD, ACM, 2006, pp. 631–636.
[16] J. Leskovec, J. Kleinberg, C. Faloutsos, Graphs over time: densification laws, shrinking diameters and possible explanations, in: KDD, ACM, 2005, pp. 177–187.
[17] J. Leskovec, A. Krevl, SNAP Datasets: stanford large network dataset collection, 2014, (http://snap.stanford.edu/data).
[18] L. Lovász, Random walks on graphs, Combinatorics, Paul Erdős is Eighty 2 (1993) 1–46.
[19] A.S. Maiya, T.Y. Berger-Wolf, Sampling community structure, in: WWW, ACM, 2010, pp. 701–710.

[20] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, E. Teller, Equation of state calculations by fast computing machines, J. Chem. Phys. 21 (6) (1953) 1087–1092.
[21] A. Pavan, K. Tangwongsan, S. Tirthapura, K.L. Wu, Counting and sampling triangles from a graph stream, in: Proceedings of the VLDB Endowment, vol. 6, 2013, pp. 1870–1881.
[22] M. Pechenizkiy, Clustering-structure representative sampling from graph streams, in: Complex Networks & Their Applications VI: Proceedings of Complex Networks 2017, vol. 689, Springer, 2017, p. 265.
[23] Y. Pei, J. Zhang, G.H. Fletcher, M. Pechenizkiy, Dynmf: Role analytics in dynamic social networks., in: IJCAI, 2018, pp. 3818–3824.
[24] U.N. Raghavan, R. Albert, S. Kumara, Near linear time algorithm to detect community structures in large-scale networks, Phys. Rev. E 76 (3) (2007) 036106.
[25] A.D. Sarma, S. Gollapudi, R. Panigrahy, Estimating pagerank on graph streams, J. ACM 58 (3) (2011) 13.
[26] L.D. Stefani, A. Epasto, M. Riondato, E. Upfal, Trièst: counting local and global triangles in fully dynamic streams with fixed memory size, TKDD 11 (4) (2017) 43.
[27] L. Tang, H. Liu, Community detection and mining in social media, Synth. Lect. Data Min. Knowl. Discov. 2 (1) (2010) 1–137.
[28] J.S. Vitter, Random sampling with a reservoir, ACM Trans. Math. Softw 11 (1) (1985) 37–57.
[29] J. Yang, J. Leskovec, Community-affiliation graph model for overlapping network community detection, in: ICDM, IEEE, 2012, pp. 1170–1175.
[30] M. Yuan, K.-L. Wu, G. Jacques-Silva, Y. Lu, Efficient processing of streaming graphs for evolution-aware clustering, in: CIKM, ACM, 2013, pp. 319–328.
[31] J. Zhang, On graph sample clustering, Eindhoven University of Technology(TU/e)[https://drive. google. com/open, Eindhoven, The Netherlands, 2018 Ph.D. thesis.
[32] J. Zhang, Y. Pei, G. Fletcher, M. Pechenizkiy, A bounded-size clustering algorithm on fully-dynamic streaming graphs, Intell. Data Anal. 22 (5) (2018) 1039–1058.
[33] J. Zhang, Y. Pei, G.H. Fletcher, M. Pechenizkiy, Structural measures of clustering quality on graph samples, in: ASONAM, 2016, IEEE, 2016, pp. 345–348.
[34] J. Zhang, Y. Pei, G. H. L. Fletcher, M. Pechenizkiy, Structural measures of clustering quality on graph samples, in: ASONAM, IEEE, 2016, pp. 345–348.
[35] J. Zhang, K. Zhu, Y. Pei, G. Fletcher, M. Pechenizkiy, Clustering affiliation inference from graph samples, in: Proceedings of the 14th International Workshop on Mining and Learning with Graphs (MLG), 2018.
[36] P. Zhao, C. Aggarwal, G. He, Link prediction in graph streams, in: ICDE, IEEE, 2016, pp. 553–564.