2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining

Streaming Graph Sampling with Size Restrictions

Anita Zakrzewska and David A Bader
School of Computational Science and Engineering
Georgia Institute of Technology, Atlanta, Georgia
Email: {azakrzewska3,bader}@gatech.edu

***Abstract*—Many graph datasets originating from online social network, financial or biological sources are too large to store or analyze. The analysis of such networks may be made more tractable if they are reduced to smaller subgraphs via sampling. While most of the known graph sampling methods are designed with static graphs in mind, many real datasets are massive and rapidly growing, making streaming methods necessary. We present two new techniques, Randomly Induced Edge Sampling (RIES) and Weighted Edge Sampling (WES). Both methods sample a stream of edges in a single pass, without the need to know future properties of the stream. In contrast to previous work that focused on limiting only the number of vertices, our methods restrict the number of edges, thus truly limiting the size of the sampled subgraph. We compare the performance of RIES and WES against the previously known streaming Random Edge (RE) method on eight social network datasets. Using four structural graph properties, we find that both RIES and WES produce subgraphs that are more structurally similar to the original graph than are the subgraphs produced by streaming RE. We also examine the sensitivity of the two algorithms with respect to their parameters. The parameters of WES affect its performance in a more predictable manner and are easier to set. Both new algorithms represent an improvement in the available streaming graph analysis toolkit.**

I. Introduction

Graphs are widely used to represent relational datasets from a variety of domains, such as online social networks, financial transactions, and biological data. In recent years, there has been increasing interest in the analysis of large, real-world networks, especially from online activity. However, many such online datasets are not only large, but constantly growing as new data is rapidly generated.Both the size and streaming nature of these graphs makes storing and analyzing them difficult. Both the size and streaming nature of these graphs make storing and analyzing them difficult. In order for analysis performed on sampled subgraphs to be useful, they should be as structurally similar to the full, unknown graphs as possible. Moreover, because these large online datasets are rapidly changing, static sampling approaches are insufficient. Therefore, in this paper we focus on streaming graph sampling: methods that can sample a subgraph with a single pass over the data stream. The single pass requirement is important in order to be able to tap into a stream of new data as it is produced and in real time make a decision about what to include and what to forget. Finally, our goal is to sample a smaller subgraph because the full dataset is too large to store or too large to run computationally intensive algorithms on. Thus, the streaming sampling method must limit the size of the sampled subgraph. This means that not only the number of vertices, but the number of desired edges can be input to the sampling algorithm.

ASONAM '17, July 31 - August 03, 2017, Sydney, Australia

ACM ISBN 978-1-4503-4993-2/17/07…$15.00
http://dx.doi.org/10.1145/3110025.3110058

A. Contributions

In this paper, we present two new algorithms for streaming graph sampling, Weighted Edge Sampling (WES) and Randomly Induced Edge Sampling (RIES). Both of these methods can (1) sample a graph stream in a single pass, thus allowing for true sampling of real data streams, and (2) limit the size of sampled subgraph by restricting the number of edges. To the best of our knowledge, the only existing algorithm that satisfies both of these requirements is streaming Random Edge Sampling (RE). Other approaches either require more than a single pass over the data (or require random access to the graph) or they only restrict the number of vertices in the sampled subgraph and not the number of edges, which means that the final size of the subgraph is not known. We test WES and RIES on graphs from several social networks and show that they produce subgraphs that are more structurally similar to the original graph compared to RE.

II. Background and Related Work

A. Sampling Goals

Graph sampling may be used to achieve four categories of goals [1]. Firstly, it may be performed to estimate specific graph parameters, such as the average degree or clustering coefficient. For example, triangle counting has received much attention in the literature [2][3]. Another goal may be to obtain representative vertices in order to measure their attributes. This may be useful, for example, in the social science setting where vertices represent people in a network with features that need to be estimated. In this case, it is most important to obtain an unbiased sample of vertices. Third, it may be performed to obtain a representative set of edges in order to analyze their attributes. For example, in a network where vertices have attributes, sampling edges can be used to measure the homophily of such attributes: do vertices tend to connect to similar vertices or not? Finally, sampling a graph can be used to obtain a smaller, but structurally similar subgraph. In this

case, simply obtaining representative vertices or representative edges is not sufficient. The goal is to obtain a representative subgraph. This is useful when the entire dataset in question is too large to either store or to analyze. Analysis may then be performed on the smaller subgraph instead. Many graphs available and used in the literature are in fact samples themselves. In this paper, we address this fourth goal of sampling: to obtain a smaller, but structurally similar subgraph.

B. Notation

We define a graph $G = (V, E)$ as a set of vertices V and edges $(u, v) \in E$. A sampling method produces a subgraph $G_S = (V_S, E_S)$ where $V_S \subseteq V$ and $E_S \subseteq E$. Typically, $V_S \subset V$ and $E_S \subset E$. In the streaming case, we have a stream of edges S and the full graph $G = (V, E)$ is creating by including all edges from S in E and all endpoint vertices of such edges in V. If the stream is never-ending, then the full graph G may constantly grow.

C. Static Graph Sampling

Most previous work on graph sampling applies only to static graphs. These methods assume that the entire dataset is available beforehand and can be accessed as needed. Some approaches assume random access ability, which may be infeasible on very large datasets, while others are more suitable to reading large, disk bound graphs. However, they all assume both that the entire dataset already exists and is not growing and that the whole graph can be stored and accessed. Static graph sampling has been studied in [4] [5] [6]. Below we describe some commonly used static sampling methods from the literature.

1) Random Edge: Edges are chosen uniformly at random from G to form the sampled graph G_S. This strategy can produce sparsely connected graphs with a larger diameter, but can easily be extended to the fully streaming context, as discussed later.

2) Random Node: In Random Node sampling, nodes are chosen uniformly at random from G. The sampled graph G_S is then formed from all edges induced by the selected vertices (edges with both endpoint vertices selected). Unlike in Random Edge sampling, the method can only target a specific number of vertices in the sample, $\mid V_S \mid$, but the resulting number of edges $\mid E_S \mid$ is not fixed and will not be known ahead of time. Thus, the final size of the graph, in terms of bytes, cannot be specified. Variations of Random Node choose the vertices not uniformly, but with some bias. These include Random Degree Node and Random PageRank Node, where the vertices are chosen with probability proportional to their PageRank score and degree, respectively. Because these methods require information about the structure of the graph prior to sampling, they are not useful for scenarios in which the full dataset is too large to process.

3) Traversal Sampling Methods: A sampled subgraph can also be obtained through graph traversal methods, which start with some initial vertices and expand by accessing neighbors. In Random Walk sampling, random walks are performed from a starting node and G_S is then formed from all the vertices encountered and edges traversed [5]. Snowball sampling performs a Bread First Search like traversal from an initial seed vertex, but only a fixed number of neighbors is added for each vertex. In Forest Fire sampling, the traversal is executed by starting with a seed vertex, visiting or "burning" a geometrically distributed random number of its neighbors, and repeating this recursively [5] [7]. Each vertex can only be "burned" or visited once.

4) Local Degree Sparsification: Lindner *et al.* [8] present Local Degree Sparsification, in which for each node v, only a fraction of neighbors with the highest degree are kept. This approach aims to select edges that lead to high degree hubs in the graph.

D. Streaming Graph Sampling

Streaming graph sampling creates and maintains a subgraph G_S by processing a stream of edges S. Each element of the stream is an edge (u, v), possibly along with some attributes such as a weight or timestamp. While the entire stream of edges together forms the full graph G, the stream may be never ending as new edge activity is generated. Imagine, for example, tapping into a stream of Twitter activity such as retweets or user follows. The full dataset is both extremely large and new data is rapidly being generated. In such a scenario, static methods cannot be applied because new data is constantly produced, while a static approach assumes we already have the entire dataset. To address this case, streaming graph sampling must be able to process a stream of edges in a single pass.

Note that some static sampling algorithms work by performing several passes over the edges of a graph. Although these methods are often called streaming, they are different from what we refer to as streaming sampling. In this work, streaming sampling algorithms are those that can process a real time stream of continuously generated data. The full dataset may never be stored so sampling must be performed in a single pass over the edge stream.

1) Reservoir Sampling: While reservoir sampling is not a graph-specific method, some streaming graph sampling algorithms in the literature use the concept of reservoir sampling [9]. This method returns a fixed size sample from a stream so that each element has an equal chance of being returned in the sample, without needing to know the total number of elements in the stream. To obtain a sample of size k, the first k elements are added to the reservoir. Each subsequent element is added with decreasing probability. The i^{th} element is added with probability k/i for $i > k$ and if added, replaces a random element of the reservoir.

In weighted reservoir sampling by Efraimidis and Spirakis, each element of the stream is included in the returned sample with probability proportional to its weight [10]. Each element with weight w is assigned a random value $u \in (0, 1)$ in order to generate a key $u^{\frac{1}{w}}$. At the end of the stream, the k elements with largest keys are returned in the sample. The same approach can also be used for unweighted sampling.

```
Data: stream S of edges, sample size k
Result: G_S = (V_S, E_S)
i = 0;
while S has edge e_i do
    (u, v) = e_i;
    if i < k then
        E_S = E_S ∪ {(u, v)};
    else
        p = k/i;
        draw r from Uniform(0, 1) if r < p then
            select random edge (q, z) ∈ E_S;
            E_S = E_S \ {(q, z)};
            E_S = E_S ∪ {(u, v)};
    i = i + 1;
for edge (u, v) ∈ E_S do
    V_S = V_S ∪ {u, v};
```

Algorithm 1: `Streaming Random Edge (RE) Sampling`

2) Random Edge Sampling: The simplest form of streaming graph sampling is Random Edge sampling (referred to throughout this paper as RE), which uses reservoir sampling to select random edges from the stream and is shown in Algorithm 1. This approach is used for structural compression of a stream in [11]. Tabassum and Gama [12] apply both RE and a streaming edge sampling technique called biased random sampling on a phone call graph. The biased random sampling method inserts each new edge into the sampled graph, replacing an old edge if the sample has reached its size limit. They find that biased random sampling yields samples that are more connected, with higher edge weights and higher average vertex degrees compared to RE. By comparing RE to biased random sampling, [12] is analyzing the effect of temporal ordering on sampling. In our work, we present general methods for streaming sampling that do not assume a certain order of the graph stream.

3) Streaming Time Node Sampling: In addition to RE, which limits the number of edges in the sampled subgraph, there exist two streaming sampling methods that limit the number of vertices. The first is Streaming Time Node Sampling by Ahmed *et. al* [13], in which the stream of edges is divided into time intervals, such as a day or hour of activity. Each interval is selected with a certain probability and all vertices that appear within a chosen interval are added to the sample. An edge from the stream is then inserted if both its endpoint vertices are in the sampled subgraph. This is similar to static node sampling with induced edges, except that in the streaming case edges are only included if they appear after both vertices are selected. Edges occurring earlier in the stream are not included. By choosing vertices that appear together within a given time interval, the authors state that the vertices are more likely to be connected. The Streaming Time Node Sampling method cannot be used to sample from a true, real-time stream of edges because it requires knowledge of the entire graph stream. In order to select a fraction ϕ of vertices in the sample, the vertices present in each unit of time are selected with probability m/T, where T is the total number of timestamps and m is the average number of timestamps needed to achieve the sampling fraction ϕ. This information is likely not present in a real streaming sampling scenario, but rather calculated once the data is collected.

4) Partially Induced Edge Sampling: The second streaming sampling method that limits the number of vertices chosen is Partially Induced Edge Sampling (PIES) [14] [1]. The PIES method is similar in principle to Streaming Time Node Sampling, but eliminates the need to know beforehand information about the entire graph stream, making it more suitable to real sampling use cases. This approach uses reservoir sampling to select a fixed number of vertices from the stream and builds a subgraph from all edges whose endpoint vertices are in the reservoir. Since edges are only included after their endpoint vertices are selected, the edges are partially, not fully, induced. Although the authors also consider an alternative in which vertices are independently chosen, the PIES method samples on edges and either rejects the edge or adds both endpoint vertices of the edge to the sample. This approach biases the sample towards higher degree vertices because they will appear on more edges and thus have a higher probability of being selected. While PIES performs well in terms of quality, it samples a subgraph with a fixed number of vertices, not a fixed number of edges. In Section III-A we discuss why this is not desirable when space is limited and the goal is to sample a fixed size subgraph.

5) Other Methods: In addition to methods that limit the number of edges or the number of vertices in the sample, some streaming graph sampling approaches do not limit the final size of the graph. One such approach is used for the DOULION algorithm for streaming triangle counting [15]. In order to reduce the amount of required computation, DOULION first samples a stream of edges before performing triangle counting. The sampling approach used selects each edge with a uniform probability. In a streaming environment in which new data is being generated, the total number of edges in the stream will not be known and this approach will therefore not restrict the final number of edges in the sample. Another example of a sampling method that does not limit the number of vertices or edges in the sample is the one used in the Graph Sample and Hold Framework [16]. There, each edge is selected with varying probability depending on how it connects to the current sample. Because edges are never removed, the size of the sample increases continually.

III. New Algorithms

A. Targeting a Vertex Versus Edge Size

Sampling methods typically create a subgraph G_S with either a specific number of vertices $\mid V_S \mid = k$ or a specific number of edges $\mid E_S \mid = k$. For example, in node based sampling, as described in Section II-C, k vertices will be randomly chosen and all edges induced by these k vertices are

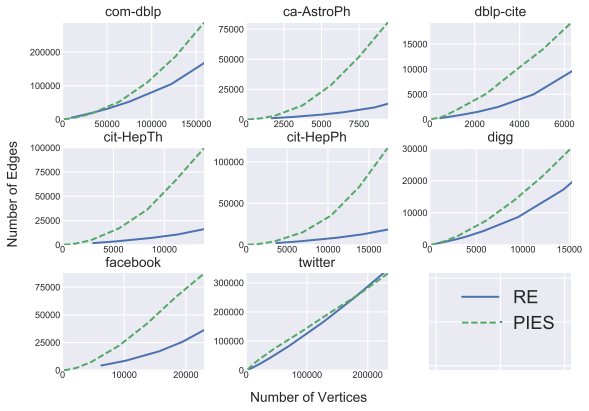

Fig. 1: The number of vertices and edges for subgraphs sampled with random edge (RE) and PIES is shown. For samples with the same number of vertices, the PIES method results in many more edges compared to the RE method.

included in E_S. The resulting number of edges $\mid E_S \mid$ in the sample cannot be predicted and can be as low as $\frac{k}{2}$ or as high as $\frac{k*(k-1)}{2}$. The same phenomenon applies to the PIES method for streaming sampling described in Section II-D. Given the fact that most social network type graphs are sparse, a bound of $\mathcal{O}(k^2)$ edges for k vertices is not relevant in practice. The resulting number of edges is especially hard to determine in streaming sampling because the dataset is not available ahead of time. The downside to this is that if the number of edges cannot be limited, then the resulting size of the subgraph G_S in bytes also cannot be limited. This is a problem when the purpose of graph sampling is to obtain a smaller subgraph because the full dataset is too large to be stored.

Therefore, when sampling with size restrictions, it is crucial to choose a method that can directly limit the number of edges and therefore the size of the graph. Note that $\mid V_S \mid \leq 2 \mid E_S \mid$ so that bounding the number of edges does bound the entire size of the graph. In cases when sampling is performed because computationally expensive analytics cannot be run on the full dataset, it is also important to limit $\mid E_S \mid$ because the running time of most algorithms will depend in part on $\mid E_S \mid$.

To further motivate the need to consider the full size of the sampled graph, we compare the number of vertices and edges in subgraphs created with PIES and RE on test graphs described in Section IV. This comparison is shown in Figure 1. Because in PIES it is not possible to target a specific number of edges, we repeatedly sample subgraphs with increasing numbers of vertices and plot the resulting number of edges against the number of vertices. Similarly, in streaming RE, it is not possible to target a specific number of vertices, so we repeatedly sample subgraphs with increasing numbers of edges and plot the number of edges against the resulting number of vertices. Figure 1 shows two important points. First, for subgraphs with a given number of vertices, those subgraphs created by PIES tend to have many more edges compared to subgraphs created by RE. This is as expected because PIES includes all induced edges going forward in time. Second, for any target number of vertices, the number of resulting edges will vary depending on the graph. Thus, it is impossible to predict $\mid E_S \mid$ based on a selected $\mid V_S \mid$ without additional knowledge about the dataset.

In this work we are concerned with streaming graph sampling. The two previous streaming methods from the literature are RE and PIES, both described in Section II-D. Because PIES limits the number of vertices, but not the number of edges, in the sampled subgraph, it does not address our use case. RE targets a specific number of edges so we use this existing method as our performance baseline. In Sections III-B and III-C, we present two new streaming sampling techniques that target a specified number of edges. In Section IV we compare these and RE on datasets from real social networks.

B. Weighted Edge Sampling (WES)

Our first streaming graph algorithm, Weighted Edge Sampling (WES), is a method that randomly samples k_e edges from a graph stream S to create a subgraph $G_S = (V_S, E_S)$. S may have any number of elements or be never ending, with edges constantly being generated. The subgraph $G_S = (V_S, E_S)$ is created by setting E_S to be all edges in the sample and V_S to be all endpoint vertices of such edges. WES uses the concept of weighted reservoir sampling to give more bias to edges in S that share vertices with the current sample G_S at the the time the edge is processed. By biasing towards such edges, WES creates a subgraph that is more connected compared to RE. Specifically, WES has two parameters, w_1 and w_2. When a new edge is processed in the stream, it is given a weight that indicates how likely it is to be included in the final sample. Edges that have no endpoint vertices in E_S when they are processed are given a weight of 1, those that have one endpoint in the sample are given a weight of w_1, and those for which both vertices are already in E_S are given a weight of w_2. A random number r is drawn from $Uniform(0, 1)$ and the key for the edge is set as $r^{1/weight}$. The k_e edges with largest keys seen so far form the sampled graph at any point in time. By setting $1 \leq w_1 \leq w_2$, WES biases towards edges that will connect to the current sampled subgraph. Full details of WES are given in Algorithm 2. In graphs with repeated edges, we sample and store each instance separately.

C. Randomly Induced Edge Sampling (RIES) Method

Our Randomly Induced Edge Sampling (RIES) is a streaming graph sampling algorithm which creates a subgraph with no more than k_v vertices and no more than k_e edges from a stream of edges S. RIES is a modification of the streaming algorithm PIES described in Section II-D. Recall that PIES randomly selects k vertices in pairs by sampling on edges and either accepting an edge and adding both its endpoint vertices to the sample or rejecting the edge. In addition, all edges in the stream whose two endpoint vertices are in the sample V_S at the time the edge arrives are added to E_S. RIES, on the other hand, includes two levels of stream sampling. On the

Data: stream S of edges, edge limit k_e, parameters w_1, w_2
Result: $G_S = (V_S, E_S)$
$V_S = \emptyset, E_S = \emptyset, i = 0$;
while S *has edge* e_i **do**
  $(u, v) = e_i$;
  if $u \in V_S$ *and* $v \in V_S$ **then**
    $weight = w_2$;
  else if $u \in V_S$ *or* $v \in V_S$ **then**
    $weight = w_1$;
  else
    $weight = 1$;
  draw r from $Uniform(0, 1)$;
  $key_i = r^{1/weight}$;
  if $\mid E_S \mid < k_e$ **then**
    $E_S = E_S \cup \{(u, v)\}$;
    $key[(u, v)] = key_i$;
    $V_S = V_S \cup \{u, v\}$;
  else if $key_i >$ *smallest key in* E_S **then**
    Remove edge (q, z) with smallest key from E_S;
    $E_S = E_S \cup \{(u, v)\}$;
    $key[(u, v)] = key_i$;
    $V_S = V_S \cup \{u, v\}$;
    if q *has no more edges in* E_S **then**
      $V_S = V_S \setminus q$;
    if z *has no more edges in* E_S **then**
      $V_S = V_S \setminus z$;
  $i = i + 1$;

Algorithm 2: `Streaming Weighted Edge Sampling (WES)`

first level, k_v vertices are sampled from the stream in pairs as in PIES. On the second level, sampling is performed on all edges in the stream whose two endpoint vertices are in the sample at the time that the edge arrives. Thus, instead of including all future induced edges, only k_e such induced edges are sampled among all those encountered in the stream. This process is shown in Algorithm 3.

Using RIES requires choosing both a maximum number of vertices k_v and maximum number of edges k_e and this choice implies an average degree $d = \frac{k_e}{k_v}$ of the resulting subgraph. This choice of d can be difficult to make because in the streaming context, the full dataset is never available and information about the degree distribution cannot be obtained. Choosing a low value of d will cause many of the induced edges that would have been included with PIES to be excluded. On the other hand, choosing too high a value of d will mean that the number of vertices k_v is too small to collect k_e edges and therefore the sampled graph will be smaller than it could have been. For experiments presented in Section IV, we choose d based on values of average degree typically seen when running PIES.

Data: stream S of edges, vertex limit k_v, edge limit k_e
Result: $G_S = (V_S, E_S)$
$V_S = \emptyset, E_S = \emptyset, i = 0, m = 0, j = 0$;
while S *has edge* e_i **do**
  $(u, v) = e_i$;
  if $\mid V_S \mid < k_v$ **then**
    $V_S = V_S \cup \{u, v\}$;
    $m = m + 1$;
  else
    $p = \frac{m}{i}$;
    draw r from $Uniform(0, 1)$;
    if $r < p$ **then**
      if $u \notin V_S$ **then**
        remove random vertex q from G_S with all incident edges;
        $V_S = V_S \cup u$;
      if $v \notin V_S$ **then**
        remove random vertex z from G_S with all incident edges;
        $V_S = V_S \cup v$;
  if $v \in V_S$ *and* $u \in V_S$ **then**
    $j = j + 1$;
    if $\mid E_S \mid < k_e$ **then**
      $E_S = E_S \cup \{(u, v)\}$;
    else
      $p = \frac{k_e}{j}$;
      draw r from $Uniform(0, 1)$;
      if $r < p$ **then**
        remove random edge (q, z) from E_S;
        $E_S = E_S \cup \{(u, v)\}$;
  $i = i + 1$;

Algorithm 3: `Randomly Induced Edge Sampling (RIES)`

TABLE I: Datasets used

Dataset	$\mid V \mid$	$\mid E \mid$
com-dblp	317,080	1,049,866
ca-AstroPh	18,771	198,050
dblp-cite	12,591	49,743
cit-HepTh	27,770	352,807
cit-HepPh	34,546	421,578
digg-reply	30,398	87,627
facebook-wall	46,952	876,993
twitter	465,017	834,797

IV. Results

A. Experimental Setup

In our problem setup, we are given a stream of edges S, which together form a graph $G = (V, E)$. Our goal in streaming graph sampling is to obtain, in a single pass over S, a subgraph $G_S = (V_S, E_S)$ that is structurally similar to G and

for which $\mid E_S \mid \leq k_e$. Of course, in a real streaming scenario new data may be constantly generated so that we never have a final full graph G. In our experiments, however, we use existing test datasets and simulate streaming so we have a full graph G to which we can compare G_S. The test graphs used are publicly available social networks from KONECT [17] and are listed in Table I.

To evaluate the quality of subgraphs created by RE, RIES, and WES, we compare how structurally similar the subgraphs created by each method are to the full graph. Specifically, we focus on the following structural properties: vertex degrees, lengths of shortest paths, clustering coefficients, and weakly connected components. To compare vertex degrees, shortest path lengths, and local clustering coefficients, we compare the distribution of these values in the full graph to that in the sampled graph. We use the two sample Kolmogorov-Smirnov (K-S) statistic to compare a distribution from G to that of G_S. The statistic is defined as $D = max_x\{\mid F_1(x) - F_2(x) \mid\}$, where F_1 and F_2 are two empirical cumulative distribution functions (CDFs). This is a standard way of comparing how structurally similar a sampled graph is to the full graph and has been used in previous work [5] [14] [1]. To compare the weakly connected components, we do not use the distribution of the sizes of connected components because many of the test graphs are close to fully connected and have few components. We cannot form a distribution from a single element and could not compare properly unless the number of connected components were large. Therefore, we evaluate connected components by computing the percentage of vertices in the largest connected component.

For results shown in Figures 2-6, we vary k_e between 0.5% and 20% of the original graph size. While results are shown in terms of the percent of edges sampled, it is important to note that WES and RIES take as input not a percentage, but a fixed number of edges k_e to include in the sample. For RIES, we use the average degree $d = 4$ and set $k_v = \frac{k_e}{d}$ accordingly. For WES, we use a $w_1 = 1$ and $w_2 = 100$. We found that biasing heavily towards edges with two endpoint vertices already in the sample created the best results. For each value of k_e, we perform 10 runs, each time randomly permuting the order of the edge stream. In Section IV-C we discuss the effect of different parameter settings of both WES and RIES.

B. Results

Figures 2, 3, and 4 show the average K-S statistic for shortest path length distributions, degree distributions, and clustering coefficient distributions plotted against sampling percentages for all test graphs. Values near zero indicate that the sampling method produces distributions that are very similar to that of the full graph, while values near one indicate dissimilar structure. For each sampling percentage shown on the y-axis, k_e is set to that percent of the total number of edges in the full graph. We plot K-S values against edge percentages instead of the actual number of edges k_e in order to make comparisons between datasets easier and to show what proportion of edges is needed for good results. However, WES, RIES, and RE all take as input a fixed number of edges to include in the sample, not a percentage.

From Figure 2, we can see that both WES and RIES produce much better results in terms of shortest path lengths compared to RE. Overall WES and RIES perform similarly, with WES performing better on some datasets and RIES on others. To understand the cause of these K-S values, we can examine the left panel of Figure 6, which shows, for the com-dblp graph, the distribution of shortest path lengths for the full graph and for samples of 10% edges using RE, WES, and RIES. At this sampling percentage, RE performs poorly because it creates a subgraph with longer shortest paths. That is, pairs of vertices tend to be at a greater distance compared to the full graph. WES and RIES, on the other hand, are able to create a subgraph with path lengths closer to those of the original graph.

For most datasets, there is a dip in the path length K-S values for RE (and sometimes RIES) at a sampling percentage of 1% or 2.5%. We will use the com-dblp graph as an example to explain this behavior. At sampling percentages of 0.5% and 1%, the subgraphs created by RE are very disconnected and thus have path lengths that are much *lower* than in the full graph, resulting in high K-S values. At 5%, however, the path lengths are typically *higher* in the sampled subgraph than in the full graph, which also results in a high K-S value. As the sampling rate increases and the path lengths change from being too short to being too long, the crossover point occurs around 2.5%. The dip in K-S values corresponds to this crossover point. Then, for all sampling rates between 5% and 20%, the subgraphs of com-dblp created by RE have shortest path lengths that are longer than those in the original graph.

Figure 3 and the middle panel of Figure 6 show results for vertex degree distributions. Again, WES and RIES tend to sample subgraphs with more similar structure to that of the full graph. This is due to the fact that both methods obtain higher vertex degrees compared to RE. WES achieves this by biasing to edges that already have endpoint vertices in the sample (thus targeting vertices with multiple edges), while RIES does so by including only edges that connect a small set of vertices.

Clustering coefficient similarity is shown in Figure 4 and the right panel of Figure 6. For most graphs, especially those with many triangles, all methods underestimate the number of triangles and therefore the local clustering coefficients. However, both RIES and WES perform better than RE, with RIES producing the best clustering results.

Finally, connected component results are plotted in Figure 5, which shows the percentage of vertices that are in the largest connected component both for the full graph and for each sampling method. Note that unlike in Figures 2, 3, and 4, the y-axis in Figure 5 does not plot either a distance or similarity metric. Good results are instead indicated by values that are close to those of the full graph. Because the test graphs we used are highly connected, the majority of vertices in the full graph are in the largest connected component. Therefore, higher values indicate better results. WES creates the most

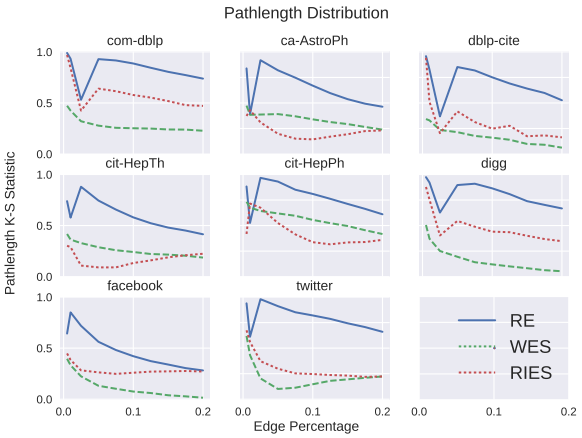

Fig. 2: For each method, the K-S statistic is used to compare the similarity of the shortest path length distribution in the sampled subgraph to that of the full, original graph. Average K-S values are shown for all percentages of edges sampled. Lower values indicate better results.

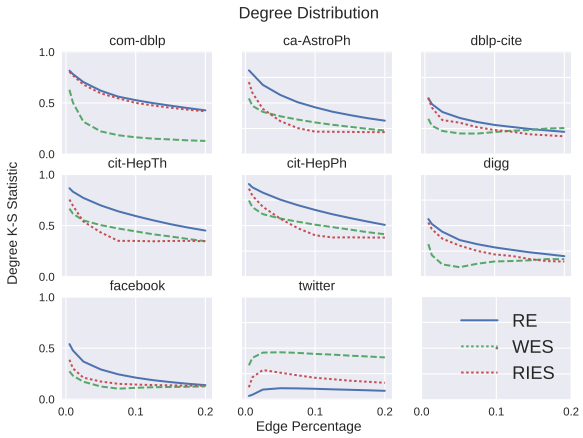

Fig. 3: For each method, the K-S statistic is used to compare the similarity of the degree distribution in the sampled subgraph to that of the full, original graph. Average K-S values are shown for all percentages of edges sampled. Lower values indicate better results.

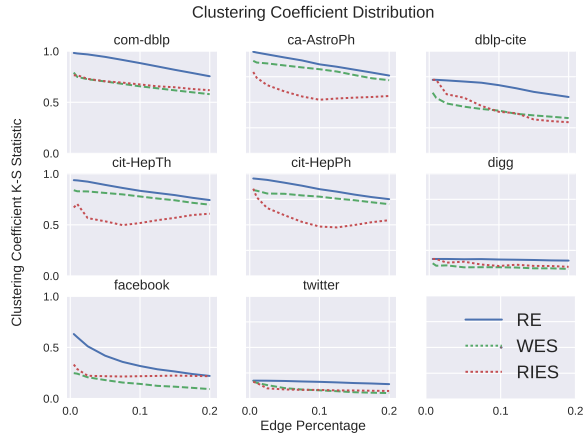

Fig. 4: For each method, the K-S statistic is used to compare the similarity of the local clustering coefficient distribution in the sampled subgraph to that of the full, original graph. Average K-S values are shown for all percentages of edges sampled. Lower values indicate better results.

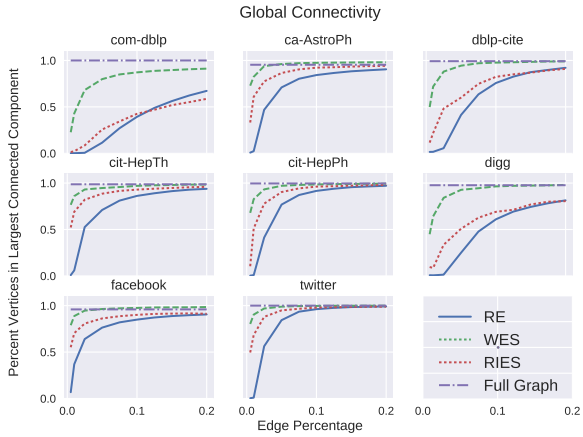

Fig. 5: The percentage of vertices in the largest connected component is plotted for both the full graph and for each sampling method for all sampling percentages. Because the test graphs used are highly connected, higher values are better.

connected samples and thus performs best, while RE creates the most disconnected graphs. The reason why WES creates more connected subgraphs than RIES is because the method automatically biases towards including edges that connect the rest of the sample. In RIES, the connectivity will depend on whether the randomly chosen vertices will tend to connect or not, so this may vary quite a bit.

C. Effect of Method Parameters

The previously discussed results for WES and RIES were obtained using a single parameter setting for each. In Figure 7, we plot shortest path length results for several different parameter settings for both WES and RIES in order to examine the effect of varying the parameters. The top plots in Figure 7 show K-S statistic values on three graphs for $w_2 = 10, 25, 50, 100, 200$, with $w_1 = 1$. Increasing the value of w_2 creates predictable results: the distribution of shortest path lengths becomes more similar to that of the full graph and the K-S statistic decreases. Similar patterns occurred for other graph measures. A higher value of w_2 created a more connected graph with higher average degrees, higher clustering coefficient, and shorter distance between pairs of vertices. Generally, this meant that a higher w_2 parameter resulted in samples more similar to the full graph. However, this need not always be the case. For example, if a graph has very low clustering coefficients or is very disconnected, a high w_2

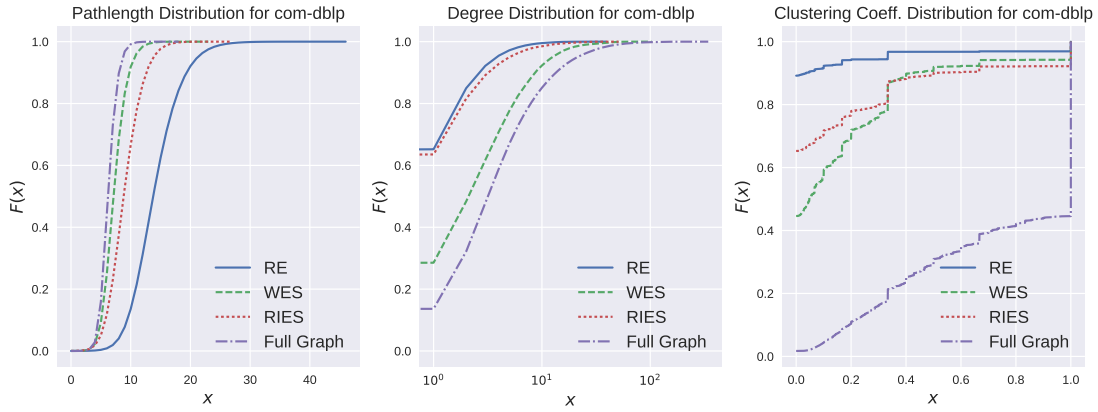

Fig. 6: This plot shows the CDFs of shortest path lengths between pairs of vertices, vertex degrees, and local clustering coefficients for the com-dblp graph. CDFs are shown for the full graph, and samples of 10% of edges using the RE, WES, and RIES methods.

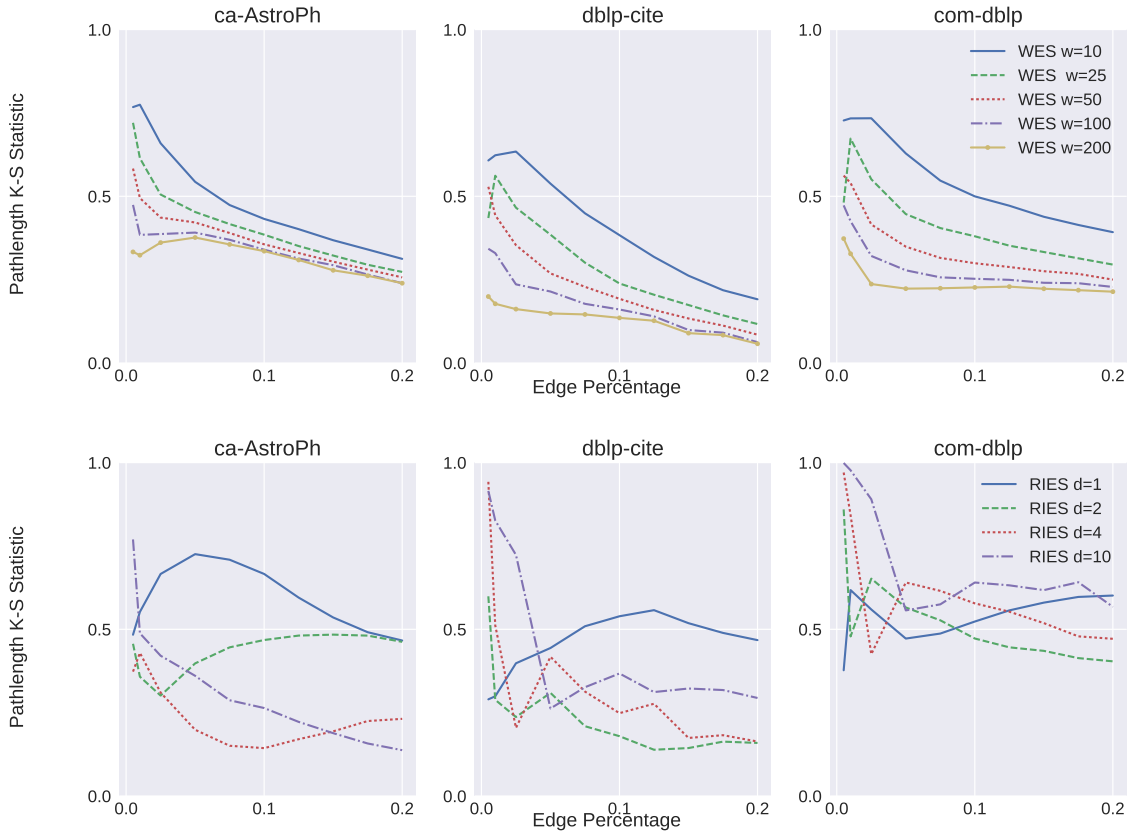

Fig. 7: The effect of varying method parameters is shown. In the top plot, the K-S statistic for shortest path length distributions is shown using three datasets and five different parameter settings of WES ($w_1 = 1$ for all and $w_2 = 10, 25, 50, 100, 200$). In the bottom plot, results are shown for four different parameter settings of RIES ($d = 1, 2, 4, 10$). The effect of a parameter change for RIES is not consistent, while for WES increasing w_2 has a predictable effect.

might create a subgraph with too many triangles and too few components. We found that the parameter w_1 had much less effect on the sample than w_2 did. The fact that the effect of increasing or decreasing the parameters of WES is predictable is important for the method to be useful in practice.

The bottom plots of Figure 7 show results for RIES with $d = 1, 2, 4, 10$. As k_e varies between 0.5% and 20% of total edges, we set $k_v = \frac{k_e}{d}$. Unlike with WES, the effect of parameter d is not predictable as it varies both across datasets and for different sampling percentages. In the streaming context, information about degree distribution will likely not be available, making it more difficult to choose a value for d. While both RIES and WES perform better than RE, the main advantage of WES over RIES is that the parameters can be set more easily and their effect is more predictable.

V. Conclusion

In this paper, we have presented two new methods for streaming graph sampling, WES and RIES. These methods sample a subgraph from an edge stream in a single pass without assuming any order of the edges and without requiring any information about the full graph. Because our goal in sampling is to obtain a smaller subgraph, our methods restrict the number of edges and not only the number of vertices. The other streaming approach from the literature that meets this requirement is RE. Through experiments on several graphs from social networks, we show that WES and RIES create sampled subgraphs that are more structurally similar to the full graph than does RE. While both methods perform well, the advantage of WES over RIES is that there is no need to set the average degree of the subgraph and the effect of the parameter setting is more predictable. The methods presented in this work process a single edge at a time and are therefore fully streaming. Future work will consider whether improved results may be obtained by using a buffer to accumulate multiple new edges before processing them.

Acknowledgement

The work depicted in this paper was sponsored in part by the National Science Foundation under award #1339745. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the National Science Foundation.

References

[1] N. K. Ahmed, J. Neville, and R. Kompella, "Network sampling: From static to streaming graphs," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 8, no. 2, pp. 7:1–7:56, 2014.

[2] A. Pavan, K. Tangwongsan, S. Tirthapura, and K.-L. Wu, "Counting and sampling triangles from a graph stream," *Proceedings of the VLDB Endowment*, vol. 6, no. 14, pp. 1870–1881, 2013.

[3] H. Jowhari and M. Ghodsi, "New streaming algorithms for counting triangles in graphs," in *International Computing and Combinatorics Conference*. Springer, 2005, pp. 710–716.

[4] S. H. Lee, P.-J. Kim, and H. Jeong, "Statistical properties of sampled networks," *Physical Review E*, vol. 73, no. 1, p. 016102, 2006.

[5] J. Leskovec and C. Faloutsos, "Sampling from large graphs," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2006, pp. 631–636.

[6] A. S. Maiya and T. Y. Berger-Wolf, "Sampling community structure," in *Proceedings of the 19th international conference on World wide web*. ACM, 2010, pp. 701–710.

[7] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graphs over time: densification laws, shrinking diameters and possible explanations," in *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*. ACM, 2005, pp. 177–187.

[8] G. Lindner, C. L. Staudt, M. Hamann, H. Meyerhenke, and D. Wagner, "Structure-preserving sparsification of social networks," in *Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2015*. ACM, 2015, pp. 448–454.

[9] J. S. Vitter, "Random sampling with a reservoir," *ACM Transactions on Mathematical Software (TOMS)*, vol. 11, no. 1, pp. 37–57, 1985.

[10] P. S. Efraimidis and P. G. Spirakis, "Weighted random sampling with a reservoir," *Information Processing Letters*, vol. 97, no. 5, pp. 181–185, 2006.

[11] C. C. Aggarwal, Y. Zhao, and S. Y. Philip, "Outlier detection in graph streams," in *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*. IEEE, 2011, pp. 399–409.

[12] S. Tabassum and J. Gama, "Sampling massive streaming call graphs," in *Proceedings of the 31st Annual ACM Symposium on Applied Computing*. ACM, 2016, pp. 923–928.

[13] N. K. Ahmed, F. Berchmans, J. Neville, and 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.

[14] N. K. Ahmed, J. Neville, and R. Kompella, "Space-efficient sampling from social activity streams," in *Proceedings of the 1st international workshop on big data, streams and heterogeneous source mining: algorithms, Systems, Programming Models and Applications*. ACM, 2012, pp. 53–60.

[15] C. E. Tsourakakis, U. Kang, G. L. Miller, and C. Faloutsos, "Doulion: counting triangles in massive graphs with a coin," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2009, pp. 837–846.

[16] N. K. Ahmed, N. Duffield, J. Neville, and R. Kompella, "Graph sample and hold: A framework for big-graph analytics," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2014, pp. 1446–1455.

[17] "The koblenz network collection KONECT," March 2017. [Online]. Available: http://konect.uni-koblenz.de