On Sampling from Massive Graph Streams

Nesreen K. Ahmed
Intel Labs
nesreen.k.ahmed@intel.com

Nick Duffield
Texas A&M University
duffieldng@tamu.edu

Theodore L. Willke
Intel Labs
ted.willke@intel.com

Ryan A. Rossi
Palo Alto Research Center
ryan.rossi@parc.com

ABSTRACT

We propose Graph Priority Sampling (GPS), a new paradigm for order-based reservoir sampling from massive graph streams. GPS provides a general way to weight edge sampling according to auxiliary and/or size variables so as to accomplish various estimation goals of graph properties. In the context of subgraph counting, we show how edge sampling weights can be chosen so as to minimize the estimation variance of counts of specified sets of subgraphs. In distinction with many prior graph sampling schemes, GPS separates the functions of edge sampling and subgraph estimation. We propose two estimation frameworks: (1) Post-Stream estimation, to allow GPS to construct a reference sample of edges to support retrospective graph queries, and (2) In-Stream estimation, to allow GPS to obtain lower variance estimates by incrementally updating the subgraph count estimates during stream processing. Unbiasedness of subgraph estimators is established through a new Martingale formulation of graph stream order sampling, in which subgraph estimators, written as a product of constituent edge estimators, are unbiased, even when computed at different points in the stream. The separation of estimation and sampling enables significant resource savings relative to previous work. We illustrate our framework with applications to triangle and wedge counting. We perform a large-scale experimental study on real-world graphs from various domains and types. GPS achieves high accuracy with $< 1\%$ error for triangle and wedge counting, while storing a small fraction of the graph with average update times of a few microseconds per edge. Notably, for billion-scale graphs, GPS accurately estimates triangle and wedge counts with $< 1\%$ error, while storing a small fraction of $< 0.01\%$ of the total edges in the graph.

1. INTRODUCTION

The rapid growth of the Internet and the explosion in online social media has led to a data deluge. A growing set of online applications are continuously generating data at unprecedented rates, from the Internet of things (*e.g.*, connected devices, routers), electronic communication (*e.g.*, email, groups, IMs, SMS), social media (*e.g.*, blogs, web pages), to the vast collection of online social networks and content sharing applications (*e.g.*, Facebook, Twitter, Youtube, Flickr). Graphs (networks) are a natural data representation in many of these application domains, where nodes represent individuals (or entities) and edges represent the interaction, communication, or connectivity among them. Consider interaction and activity networks formed from electronic communication between users (*e.g.*, emails, SMS, IMs, etc). These resulting interaction and activity networks manifest as a *stream of edges*, where edges (*i.e.*, interactions) occur one at a time, carrying a wealth of behavioral, community, and relationship information. Many of these networks are massive in size, due to the prolific amount of activity data (*e.g.*, 4.75 billion pieces of content shared daily on Facebook).

Modeling and analyzing these massive and dynamic interaction graphs have become important in various domains. For example, detecting computer/terrorist attacks and anomalous behavior in computer networks and social media [6, 1], identifying the behavior and interests of users in online social networks (e.g., viral marketing, online advertising) [22, 35], real-time monitoring and detecting virus outbreaks in human contact networks [23], among many others. But the volume and velocity of these graphs outpaces practitioners' ability to analyze and extract knowledge from them. As a result, a common practice is to analyze static windowed snapshots of these graphs over time. However, this is costly and inefficient both in terms of storage volumes, and management for future use.

To keep up with the growing pace of this data, we need efficient methods to analyze dynamic interaction networks as the data arrives in streams, rather than static snapshots of graphs. In various application domains, graph mining is rapidly shifting from mining static graph snapshots to mining an open-ended graph stream of edges representing node interactions. We would like to have a framework capable of operating continuously and efficiently, processing edges/links as they arrive and providing timely answers for various network analysis tasks. This motivates the streaming graph model in which the graph is presented as a stream of edges/links in any arbitrary order, where each edge can be processed only once, and any computation uses a small memory footprint (*i.e.*, often sublinear in the size of the input stream) [26, 25, 3].

Other scenarios require efficient methods to analyze static graphs that are too large to fit in memory [30, 26]. In the case of memory constraints, traditional graph methods are costly as they require random disk accesses that incur large I/O costs. This naturally leads to the question: how can we process graphs sequentially (one edge at a time). The streaming graph model provides an ideal framework for dynamic and naturally streaming graph data. In addition, it would also apply to the case of graph data that is stored as a list of edges streaming from storage [30, 26, 3].

Despite the recent advances in high-performance graph analysis tools and the availability of computational resources on the cloud, running brute-force graph analytics is usually too costly, too ineffi-

Proceedings of the VLDB Endowment, Vol. 10, No. 11

cient, and too slow in practical scenarios. In many cases, the cost of performing the exact computation is often not worth the extra accuracy. While an approximate answer to a query or an analysis task is usually sufficient, in particular when the approximation is performed with sufficient high-quality, unbiasedness, and confidence guarantees. Sampling provides an attractive approach to quickly and efficiently find an approximate answer to a query, or more generally, any analysis objective. While previous work on sampling from graph streams focused on sampling schemes for the estimation of certain graph properties (*i.e.*, in particular triangles) [20, 27, 8], in this paper however, we focus on an adaptive general purpose framework for sampling from graph streams. From a high-volume stream of edges, the proposed framework maintains a generic sample of limited size that can be used at any time to accurately estimate the total weight of arbitrary graph subsets (*i.e.*, triangles, cliques, stars, subgraphs with particular attributes). To obtain accurate estimates of various graph properties, we maintain a *weight sensitive* sample that devotes sampling resources to edges that are informative for those properties. In addition, we want a sampling scheme that is capable of utilizing auxiliary information about the items in the stream. Most previous work on stream sampling is either hard to adapt to various estimation objectives, focused on specific graph properties, or does not utilize auxiliary information.

Contributions. The main contributions of this paper are as follows.

- *Framework.* We propose *graph priority sampling* (GPS), the first adaptive, general purpose, weight sensitive, one-pass, fixed-size without replacement sampling framework for massive graph streams. GPS provides a general way to weight edge sampling according to auxiliary/size variables to estimate various graph properties (Sec 3). We discuss antecedents to our approach in Sec 2.
- *Theoretical Analysis.* We propose a new Martingale formulation for subgraph count estimation, and show how to compute unbiased estimates of arbitrary subgraph counts from the sample at any point during the stream. This *Post-Stream Estimation* can be used to construct reference samples for retrospective graph queries (Sec 3).
- *In-Stream Estimation* is a second framework in which subgraph count estimates are incrementally updated during stream processing rather than computed at a selected point, which can be used to obtain accurate estimates with lower variance (Sec 5).
- *Algorithms.* We design efficient/parallel algorithms for triangle and wedge count estimation using GPS (Sec 4–5).
- *Accuracy.* We test our framework on graphs from various domains and types. Our estimates are accurate with $\leq 1\%$ error. Notably, for billion-scale graphs, GPS accurately estimates triangle and wedge counts with $< 1\%$ error, while storing a small fraction of $< 0.01\%$ of the total number of edges in the graph.
- *Real-time Tracking.* The proposed framework can maintain accurate real-time estimates while the stream is evolving (Sec 6).

We survey related work in Section 7 before concluding in Section 8. Proofs of the Theorems are deferred to Section 9.

2. ANTECEDENTS TO OUR WORK

We now discuss how our proposed framework generalizes and exploits the properties of many known sampling schemes: reservoir sampling, probability proportional to size, and order sampling.

Reservoir Sampling is a class of single-pass schemes to sample a fixed number n of items from a stream of $N > n$ items [21, 33]. The sample is maintained incrementally over the stream, and can be used at any time to estimate the stream properties up to that time. Graph priority sampling uses reservoir sampling to form a fixed-size weighted sample from an input edge stream in one pass.

Probability Proportional to Size Sampling. In many real-world applications, each item has an auxiliary variable (a size or weight). Auxiliary variables correlated with the population variable under study can be used as weights for non-uniform sampling. Variants of this scheme have been designed to fulfill different estimation goals [32]. Inclusion Probability Proportional to Size (IPPS) is a variance minimizing scheme for a given average sample size [15].

Priorty Sampling is an IPPS order-based reservoir sampling with item i assigned a priority w_i/u_i, where the u_i are IID uniform on $(0, 1]$. A priority sample takes the n highest priority items, each with an unbiased estimator $\widehat{w}_i = \max\{w_i, z\}$ of w_i, where z is the $(n+1)^{st}$ highest priority. See [16, 28, 17, 11] for further details.

Most of these methods suit mainly sampling IID data (streams of database transactions, IP network packets). In this paper, however, we deal with graph data that exhibit both structure and attributes. A few of the methods discussed above have been extended to graph streams, in particular uniform-based reservoir sampling (see Section 7). Graph priority sampling generalizes most of the above sampling schemes, and obtains an adaptive, weight sensitive, general purpose, fixed-size sample in one-pass, while including topological information that we wish to estimate as auxiliary variables.

3. GRAPH PRIORITY SAMPLING

This section establishes a methodological framework for graph priority sampling. Sec 3.1 sets up our notation and estimation goals. Sec 3.2 specifies our algorithm and its properties. Sec 3.3 and 3.4 establishes the unbiasedness for our subgraph estimators and their variance. Sec 3.5 shows how to choose sampling weight to minimize the variance for target subgraphs.

3.1 Proposed Framework

Notation and Problem Definition. Let $G = (V, K)$ be a graph with no self loops, where V is the set of nodes, and K is the set of edges. For any node $v \in V$, let $\Gamma(v)$ be the set of neighbors of node v and so $\deg(v) = |\Gamma(v)|$ is the degree of v. We call two edges $k, k' \in K$ adjacent, $k \sim k'$, if they join at some node, *i.e.*, $k \cap k' \neq \emptyset$ [2]. In this paper, we are principally concerned with estimating the frequency of occurrence of certain subgraphs of G. Our proposed *graph stream model* comprises an input graph $G = (V, K)$ whose edges arrive for sampling in any arbitrary order [2]. We assume edges are unique and so we can identify each edge in K with its arrival order in $[|K|] = \{1, 2, \ldots, |K|\}$. Due to the identity between edges and arrival order we will use the notation $J \subset K$ and $J \subset [|K|]$ interchangeably. Thus, we can uniquely identify a subgraph $J \in \mathcal{J}$ with the corresponding ordered subset of edges $J \subset [|K|]$, written as an ordered subset, $J = (i_1, i_2, ..., i_\ell)$ with $i_1 < i_2 < ... < i_\ell$ being the arrival order. Thus, $J \subset [t]$ if all the edges in J have arrived by time t.

We use the general notation $\mathcal{J}$ to denote the set of all subgraphs of G whose count $N(\mathcal{J}) = |\mathcal{J}|$ we wish to estimate. As special cases, $\triangle$ will denote the set of triangles and Λ the set of wedges (paths of length 2) in G, Let $\alpha = 3N(\triangle)/N(\Lambda)$ denote the global clustering coefficient of G. For a set of subgraphs $\mathcal{J}$ we shall use the notation $\mathcal{J}_t = \{J \in \mathcal{J} : J \subset [t]\}$ to denote those members J of $\mathcal{J}$ all of whose edges have arrived by time t, The number of these is denoted $N_t(\mathcal{J}) = |\mathcal{J}_t|$

Algorithm and Intuition. The basic outline and intuition of the proposed framework comprises of two steps. In the first step, we select a small sample of edges $\widehat{K}_t \subset [t]$ from the set of all edges arriving by time t, with $m = |\widehat{K}_t|$ is the reservoir capacity. The second step allows us to estimate the count of general subgraphs

in G regardless of whether they were all sampled. We define the **subset indicator** of a subset of edges $J \subset [|K|]$ by the function,

$$S_{J,t} = \begin{cases} 1, & J \subset [t] \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

Thus, $S_{J,t} = 1$ if and only if all the edges in J have arrived by time t. In the above notation $S_{j,t} = N_t(\{j\})$ and $N_t(\mathcal{J}) = \sum_{J \in \mathcal{J}} S_{J,t}$ is the count of all members of $\mathcal{J}$ (*i.e.*, subgraphs $J \in \mathcal{J}$) whose edges have all arrived by time t. Our goal is to estimate $N_t(\mathcal{J})$ from a selected sample of edges $\widehat{K}_t \subset [t]$.

3.2 Algorithm Description & Properties

We formally state our main algorithm GPS(m) for Graph Priority Sampling into a reservoir $\widehat{K}$ of capacity m in Algorithm 1. The main algorithm GPS(m) (see Algorithm 1) maintains a dynamic reservoir/sample $\widehat{K}$ of size m from a graph whose edges are streaming over time. When a new edge k arrives (Line 3), we call the procedure GPSUPDATE. We assume a weight function $W(k, \widehat{K})$ that expresses the sampling weight for edge k as a function of both k and the topology of the reservoir $\widehat{K}$ (Line 8). For example, $W(k, \widehat{K})$ could be set to the number of sampled edges adjacent to k, or the number of triangles in $\widehat{K}$ completed by k. In general, the function $W(k, \widehat{K})$ can be set to include topology, attributes, and/or any auxiliary information in the graph. Once the weight $w(k) = W(k, \widehat{K})$ is computed, we assign edge k a priority $r(k) = w(k)/u(k)$ (Line 9), where $u(k)$ is an independent uniformly generated random number (Line 7). GPS maintains a *priority queue* where each edge in the reservoir $\widehat{K}$ is associated with a priority (computed at arrival time) that defines its position in the queue. When a new edge k arrives in the stream (and if the reservoir is full, see Lines 11–14), its priority is computed and compared with the lowest priority edge in the queue. If edge k has a lower priority, then it is discarded. If edge k has a higher priority, then the lowest priority edge is discarded and replaced by edge k.

Implementation and data structure. We implement the priority queue as a *min-heap* [12], where each edge has a priority less than or equal to its children in the heap, and the root position points to the edge with the lowest priority. Thus, access to the lowest priority edge is performed in $O(1)$. If edge k has a higher priority than the root, edge k is initially inserted at the root position, then moved downward to its correct position in the heap in $O(\log m)$ time (worst case). Note that if the sample size is less than the reservoir capacity, *i.e.*, $|\widehat{K}| < m$, edge k is inserted in the next available position in the heap, then moved upward to its correct position in the heap in $O(\log m)$ time (worst case). The threshold z^* is the $(m+1)^{\text{st}}$ highest priority (see Line 13). To simplify the analysis, we provisionally admit a new edge k to the reservoir, then one of the $m+1$ edges is discarded if it has the lowest priority. Finally, at any time in the stream, we can call the procedure GPSNORMALIZE to obtain the edge sampling probabilities. As shown in the proof of Theorem 1, $p(k') = \min\{1, w(k')/z^*\}$ (see Lines 16–17) is the conditional sampling probability for k' given z^*; hence $1/p(k')$ forms the Horvitz-Thompson estimator for the indicator of k'.

Algorithm Properties. Graph Priority Sampling demonstrates the following properties:

(S1) Fixed Size Sample. As seen above, $\widehat{K}_t$ is a reservoir sample of fixed size $|\widehat{K}_t| = m$ for all $t \geq m$.

(S2) Unbiased Subgraph Estimation. In Section 3.3 we construct unbiased subgraph estimators $\widehat{S}_{J,t}$ of $S_{J,t}$ for each subgraph J and $t > 0$. The $S_{J,t}$ is computable from the sample sets $\widehat{K}_t$. Section 5 extends our construction to new classes of estimators $\prod_{i \in J} \widehat{S}_{i,t_i}$ that are edge products over multiple times. These allow unbiased estimation of subgraphs in new ways: as they arise in the sample, or prior to discard, or on arrival of certain edges. These results follow from a novel Martingale formulation of graph priority sampling.

Algorithm 1 Family of Graph Priority Sampling Algorithms

```
1  procedure GPS(m)
2      K̂ ← ∅; z* ← 0
3      while new edge k do
4          GPSUPDATE(k, m)
5      GPSNORMALIZE(K̂)

6  procedure GPSUPDATE(k, m)
7      Generate u(k) ~ Uni(0, 1]
8      w(k) ← W(k, K̂)
9      r(k) ← w(k)/u(k)                          ▷ Priority of edge k
10     K̂ ← K̂ ∪ {k}                               ▷ Provisionally include edge k
11     if |K̂| > m then
12         k* ← arg min_{k' ∈ K̂} r(k')            ▷ Lowest priority edge
13         z* ← max{z*, r(k*)}                    ▷ New threshold
14         K̂ ← K̂ \ {k*}                           ▷ Remove lowest priority edge

15 procedure GPSNORMALIZE(K̂)
16     for k' ∈ K̂ do
17         p(k') ← min{1, w(k')/z*}               ▷ HT Renormalize
```

(S3) Weighted Sampling and Variance Optimization. Graph priority sampling provides a mechanism to tune sampling weights to the needs of applications. We accomplish this using edge weights that express the role of an edge in the sampled graph (see Sec. 3.5). Examples include the number of edges in the currently sampled graph that are adjacent to an arriving edge, and the number of subgraphs bearing a given motif that would be created by inclusion of the edge in the sample. In addition, weights may also express *intrinsic* properties that do not depend explicitly on the graph structure. Examples include endpoint node/edge identities, attributes, and other auxiliary variables, *e.g.* user age, gender, interests, etc.

(S4) Computational Feasibility. For each incoming edge $k = (v_1, v_2)$, the GPS framework calls GPSUPDATE to update reservoir $\widehat{K}$ of capacity m. The processing time per arrival comprises the cost to compute the weight (*i.e.*, $W(k, \widehat{K})$ and the cost to update the heap (if the new edge is inserted). The worst case cost of heap insertion/deletion is $O(\log m)$.

The cost of $W(k, \widehat{K})$ is problem-specific and depends on the sampling objective and the function that computes the sampling weight for edge k. We use the number of triangles in $\widehat{K}$ completed by k, *i.e.*, $W(k, \widehat{K}) = |\widehat{\Gamma}(v_1) \cap \widehat{\Gamma}(v_2)|$. This can be achieved in $O(\min\{\deg(v_1), \deg(v_2)\})$, if a hash table or a bloom filter is used for storing $\widehat{\Gamma}(v_1), \widehat{\Gamma}(v_2)$ and looping over the sampled neighborhood of the vertex with minimum degree and querying the hash table of the other vertex. The space requirements of the proposed framework GPS are: $O(|\widehat{V}| + m)$, where $|\widehat{V}|$ is the number of nodes in the reservoir, and m is the reservoir capacity. There is a trade-off between space and time, and GPS could limit the space to $O(m)$. However, the cost update per edge would require a pass over the reservoir ($O(m)$ in the worst case). On the other hand, increasing the space to $O(|\widehat{V}| + m)$ can yield sub-linear time for edge updates.

3.3 A Martingale for Subgraph Counting

We now axiomatize the dependence of w on k and $\widehat{K}$ and analyze the statistical properties of estimates based on the sample set. We index edge arrivals by $t \in \mathbb{N}$, and let $\widehat{K}_t \subset [t]$ denote the set of indices in the sample after arrival t has been processed and $\widehat{K}'_t = \widehat{K}_{t-1} \cup \{t\}$ denote the index set after t has been provision-

ally included in the sample. Let w_t denote the weight assigned to arrival t and u_t be IID uniform on $(0, 1]$. The priority of arrival t is then $r_t = w_t/u_t$. An edge $i \in \widehat{K}'_t$ is selected if it does not have the smallest priority in $\widehat{K}'_t$, i.e., if

$$r_i > z_{i,t} = \min_{j \in \widehat{K}'_t \setminus \{i\}} r_j \tag{2}$$

When i is selected from $\widehat{K}'_t$ then $r_{i,t}$ is equal to the unrestricted minimum priority $z_t = \max_{j \in \widehat{K}'_t} r_j$ since the discarded edge takes the minimum. For $t < i$, $z_{i,t} = z_t$ since i has not yet appeared. Defining $B_i(x) = \{r_i > x\}$, we write the event that i is in the sample at time $t \geq i$ as

$$\{i \in \widehat{K}_t\} = \cap_{i=s}^{t} B_i(z_{i,s}) \tag{3}$$

We now construct for each edge i a sequence of **Edge Estimators** $\widehat{S}_{i,t}$ that are unbiased estimators of the corresponding edge indicators. We prove unbiasedness by establishing that the sequence is a Martingale [34]. A sequence of random variables $\{X_t : t \in \mathbb{N}\}$ is a *Martingale* with respect to a filtration $\mathcal{F} = \{\mathcal{F}_t : t \in \mathbb{N}\}$ of increasing σ-algebra (these can be regarded as variables for conditioning) if each X_t is measurable w.r.t. $\mathcal{F}_t$ (i.e. it is function of the the corresponding variables) and obeys: $\mathbb{E}[X_t|\mathcal{F}_{t-1}] = X_{t-1}$. Martingales provide a framework within which to express unbiased estimation in nested sequences of random variables.

We express GPS within this framework. For $J \subset \mathbb{N}$ let $z_{J,t} = \min_{j \in \widehat{K}'_t \setminus J} r_j$. Let $\mathcal{F}^{(0)}_{i,t}$ denote the σ-algebra generated by the variables $\{B_j(z_{\{ij\},s}) : j \neq i,\ s \leq t\}$, let $\mathcal{F}_{i,t}$ be the σ-algebra generated by $\mathcal{F}^{(0)}_{i,t}$ and the variables $\mathcal{Z}_{i,t} = \{z_{i,s} : s \leq t\}$ and $\{B_i(z_{i,t}), i \leq s \leq t\}$, and let $\mathcal{F}_i$ be the filtration $\{\mathcal{F}_{i,t} : t \geq i-1\}$. Set $z^*_{i,t} = \max_{i \leq s \leq t} z_{i,s}$ and define:

$$R_{i,t} = \min\{1, w_i/z^*_{i,t}\}, \quad \widehat{S}_{i,t} = I(B_i(z^*_{i,t}))/R_{i,t} \tag{4}$$

for $t \geq i$ and $\widehat{S}_{i,t} = 0$ for $0 \leq t < i$.

THEOREM 1 (EDGE ESTIMATION). *Assume $w_{i,t}$ is $\mathcal{F}_{i,t-1}$-measurable. Then $\{\widehat{S}_{i,t} : t \geq i\}$ is a Martingale w.r.t. the filtration $\mathcal{F}_i$, and hence $\mathbb{E}[\widehat{S}_{i,t}] = S_{i,t}$ for all $t \geq 0$.*

The measurability condition on w_i means that it is determined by the previous arrivals, including the case that an edge weight depends on the sampled topology that it encounters on arrival.

To compute $\widehat{S}_i$ we observe that when $i \in \widehat{K}_t$, then (i) $r_i > z_{i,t}$ and hence $z_{i,t} = z_t$; and (ii) $z_{i,i} > \max_{s \leq i} z_{i,s}$ since otherwise the minimizing j for $z_{i,i}$ could not have been selected. Hence for $t \geq i$ with $z^*_t = \max_{s \leq t} z_s$:

$$\widehat{S}_{i,t} = I(i \in \widehat{K}_t)/\min\{1, w_i/z^*_t\} \tag{5}$$

It is attractive to posit the product $\widehat{S}_J = \prod_{i \in J} \widehat{S}_i$ as a subgraph estimator J when $J \subset [t]$. While this estimator would be unbiased for independent sampling, the constraint of fixed-size introduces dependence between samples and hence bias of the product. For example, VarOpt sampling [10] obeys only $\mathbb{E}[\prod_{i \in J} \widehat{S}_i] \leq \prod_{i \in J} \mathbb{E}[\widehat{S}_i]$. We now show that the edge estimators for Graph Priority Sampling, while dependent, have zero correlation. This is a consequence of the property, that we now establish, that the edge product estimator is a Martingale.

Fix a subgraph as $J \subset \mathbb{N}$, set $J_t = J \cap [t]$ and for $k \in [t] \cap J^c$ define $z_{J,k,t} = \max_{i \in \widehat{K}'_t \setminus (J_t \cup \{k\})} r_i$, i.e., the maximal rank in $\widehat{K}'_t$ apart from k and those in J_t. Let $\mathcal{F}^{(0)}_{J,t}$ denote the conditioning w.r.t $\{B_k(z_{J,k,s}) : k \notin J, s \leq t\}$, and let $\mathcal{F}_{J,t}$ denote conditioning w.r.t $\mathcal{F}^{(0)}_{J,s}$, $\mathcal{Z}_{J,t} = \{z_{J,s} : s \leq t\}$ and $\{B_i(z_{J,s}) : i \in J, s \leq t\}$ and let $\mathcal{F}_J$ denote the corresponding filtration.

THEOREM 2 (SUBGRAPH ESTIMATION). *(i) For $J \subset \mathbb{N}$ define $S_{J,t} = \prod_{i \in J} \widehat{S}_{i,t}$. Then $\{\widehat{S}_{J,t} : t \geq \max J\}$ is a Martingale w.r.t. the filtration $\mathcal{F}_J$ and hence $\mathbb{E}[\prod_{i \in J} \widehat{S}_{i,t}] = S_{J,t}$ for $t \geq 0$.*

(ii) For any set $\mathcal{J}$ of subgraphs of G, $\widehat{N}_t(\mathcal{J}) = \sum_{J \in \mathcal{J}: J \subset K_t} \widehat{S}_{J,t}$ is an unbiased estimator of $N_t(\mathcal{J}) = |\mathcal{J}_t| = \sum_{J \in \mathcal{J}} S_{J,t}$, and the sum can be restricted to those $J \in \mathcal{J}$ for which $J \subset \widehat{K}_t$, i.e., entirely within the sample set at t.

The proof of Theorem 2(i) mirrors that of Theorem 1, and follows from the fact that the expectation of $\prod_{j \in J} B_j(z_{J,t})$, conditional on $\mathcal{F}_{J,t-1}$, is a product over J; we omit the details. Part (ii) follows by linearity of expectation, and the restriction of the sum follows since $\widehat{S}_{J,t} = 0$ unless $J \subset \widehat{K}_t$.

3.4 Variance and Covariance Estimation

Theorem 2 also allows us to establish unbiased estimators for the variance and covariance amongst subgraph estimators. Consider two edge subsets $J_1, J_2 \subset \widehat{K}_t$. We use the following as an estimator of $\mathrm{Cov}(\widehat{S}_{J_1,t}, \widehat{S}_{J_2,t})$:

$$\begin{aligned}\widehat{C}_{J_1,J_2,t} &= \widehat{S}_{J_1,t}\widehat{S}_{J_2,t} - \widehat{S}_{J_1 \setminus J_2,t}\widehat{S}_{J_2 \setminus J_1,t}\widehat{S}_{J_1 \cap J_2,t} \\ &= \widehat{S}_{J_1 \setminus J_2,t}\widehat{S}_{J_2 \setminus J_1,t}\widehat{S}_{J_1 \cap J_2,t}\Big(\widehat{S}_{J_1 \cap J_2,t} - 1\Big) \\ &= \widehat{S}_{J_1 \cup J_2,t}\Big(\widehat{S}_{J_1 \cap J_2,t} - 1\Big) \end{aligned} \tag{6}$$

THEOREM 3 (COVARIANCE ESTIMATION). *$\widehat{C}_{J_1,J_2,t}$ is an estimator of $\mathrm{Cov}(\widehat{S}_{J_1,t}, \widehat{S}_{J_2,t})$.*

(i) $\widehat{C}_{J_1,J_2,t}$ is an unbiased estimator of $\mathrm{Cov}(\widehat{S}_{J_1,t}, \widehat{S}_{J_2,t})$.

(ii) $\widehat{C}_{J_1,J_2,t} \geq 0$ and hence $\mathrm{Cov}(\widehat{S}_{J_1,t}, \widehat{S}_{J_2,t}) \geq 0$.

(iii) $\widehat{S}_{J,t}(\widehat{S}_{J,t} - 1)$ is an unbiased estimator of $\mathrm{Var}(\widehat{S}_{J,t})$.

(iv) $\widehat{C}_{J_1,J_2,t} = 0$ if and only if $\widehat{S}_{J_1,t} = 0$ or $\widehat{S}_{J_2,t} = 0$, or $J_1 \cap J_2 = \emptyset$, i.e., covariance estimators are computed only from edge sets that have been sampled and their intersection is non-empty.

We do not provide the proof since these results are a special case of a more general result that we establish in Section 5 product from graph estimators in which the edges are sampled at different times.

3.5 Optimizing Subgraph Variance

How should the ranks $r_{i,t}$ be distributed in order to minimize the variance of the unbiased estimator $\widehat{N}_t(\mathcal{J})$ in Theorem 2 ? This is difficult to formulate directly because the variances of the $\widehat{S}_{j,t}$ cannot be computed simply from the candidate edges. Instead, we minimize the conditional variance of the increment in $N_t(\mathcal{J})$ incurred by admitting the new edge to the sample: more precisely:

1. For each arriving edge i find the marginal selection probability for i that minimizes the conditional variance $\mathrm{Var}(\widehat{N}_i(\mathcal{J})|\mathcal{F}_{i,i-1})$.
2. Edges are priority order sampled using weights that implement the variance-minimizing selection probabilities.

Our approach is inspired by the cost minimization approach of IPPS sampling [15]. When $i \in \widehat{K}'_t$ we define $\widehat{N}_{i,t}(\mathcal{J}) = \#\{J \in \mathcal{J} : i \in J \wedge J \subset \widehat{K}'_t\}$ i.e., the number of members J of $\mathcal{J}$ that are subsets of the set of candidate edges $\widehat{K}'_t$ and that contain i. Put another way, $\widehat{N}_{i,t}(\mathcal{J})$ is the number of members of $\mathcal{J}$ that are created by including i within $\widehat{K}_t$. Suppose i is sampled with probability p, conditional on $\mathcal{F}_{i,i-1}$. The expected space cost of the

sample is proportional to p, while the sampling variance associated with Horvitz-Thompson estimation of the increment $n = \widehat{N}_{i,t}(\mathcal{J})$ is $n^2(1/p - 1)$. Following [15], we form a composite cost

$$C(z) = z^2 p + n^2(1/p - 1) \quad (7)$$

where z is a coefficient expressing the relative scaling of the two components in the cost. $C(z)$ is minimized by $p = \min\{1, n/z\}$, corresponding to IPPS sampling with threshold z and weight n. By comparison with the relation between threshold sampling and priority sampling, this suggests using $n = \widehat{N}_{i,t}(\mathcal{J})$ as the weight for graph stream order sampling. We also add a default weight for each edge so that an arriving edge k that does not currently intersect with the target class (i.e. $k \neq J \subset \widehat{N}_t(\mathcal{J})$) can be sampled.

4. TRIANGLE & WEDGE COUNTING

In this section we apply the framework of Section 3 to triangle and wedge counting, and detail the computations involved for the unbiased subgraph count estimates and their variance estimates.

Unbiased Estimation of Triangle/Wedge Counts. From the notation in Sec. 3, let $\triangle_t$ be the set of triangles whose edges have arrived by t, and $\widehat{\triangle}_t \subset \triangle_t$ be the subset of such triangles that appear in the sample $\widehat{K}_t$. Then, $\widehat{N}_t(\triangle)$ is the Horvitz-Thompson estimator of the count of members (*i.e.*, triangles) in $\triangle_t$. We write $\tau \in \triangle_t$ as a subset (k_1, k_2, k_3) ordered by edge arrival (*i.e.*, k_3 is the last edge). Similarly, Λ_t denotes wedges whose edges have arrived by time t. So $\widehat{N}_t(\Lambda)$ is the Horvitz-Thompson estimator of the count of wedges in Λ_t, and $\lambda \in \Lambda_t$ is written as an ordered subset (k_1, k_2) with k_2 the last edge. The following are direct corollaries of Theorem 2:

COROLLARY 1 (TRIANGLE COUNT). $\widehat{N}_t(\triangle) = \sum_{\tau \in \triangle_t} \widehat{S}_{\tau,t}$ *is an unbiased estimator of* $N_t(\triangle)$.

COROLLARY 2 (WEDGE COUNT). $\widehat{N}_t(\Lambda) = \sum_{\lambda \in \Lambda_t} \widehat{S}_{\lambda,t}$ *is an unbiased estimator of* $N_t(\Lambda)$.

Additionally, we use $\widehat{\alpha}_t = 3\widehat{N}_t(\triangle)/\widehat{N}_t(\Lambda)$ as an estimator for the global clustering coefficient α_t.

Variance Estimation of Triangle/Wedge Counts. Let $\mathrm{Var}[\widehat{N}_t(\triangle)]$ denote the variance of the unbiased estimator of triangle count at time t, and $\mathrm{Var}[\widehat{N}_t(\Lambda)]$ the variance of the unbiased estimator of wedge count at time t, given by Corollaries 1 and 2 respectively. Expressions for unbiased estimates of these variances are direct corollaries from Theorem 3, which itself follows from Theorem 5.

COROLLARY 3 (VARIANCE OF TRIANGLE COUNTS). $\widehat{V}_t(\triangle)$ *is an unbiased estimator of* $\mathrm{Var}[\widehat{N}_t(\triangle)]$*, where*

$$\widehat{V}_t(\triangle) = \sum_{\tau \in \triangle_t} \widehat{S}_{\tau,t}(\widehat{S}_{\tau,t} - 1) + 2 \sum_{\tau \in \triangle_t} \sum_{\substack{\tau' < \tau \\ \tau' \in \triangle_t}} \widehat{C}_{\tau,\tau',t} \quad (8)$$

COROLLARY 4 (VARIANCE OF WEDGE COUNTS). $\widehat{V}_t(\Lambda)$ *is an unbiased estimator of* $\mathrm{Var}[\widehat{N}_t(\Lambda)]$*, where*

$$\widehat{V}_t(\Lambda) = \sum_{\lambda \in \Lambda_t} \widehat{S}_{\lambda,t}(\widehat{S}_{\lambda,t} - 1) + 2 \sum_{\lambda \in \Lambda_t} \sum_{\substack{\lambda' < \lambda \\ \lambda' \in \Lambda_t}} \widehat{C}_{\lambda,\lambda',t} \quad (9)$$

Variance Estimation for Global Clustering Coefficient. We use $\widehat{\alpha} = 3\widehat{N}_t(\triangle)/\widehat{N}_t(\Lambda)$ as an estimate of the global clustering coefficient $\alpha = 3N(\triangle)/N(\Lambda)$. While this estimate is biased, asymptotic convergence to the true value for large graphs would follow from the property of $\widehat{N}_t(\triangle)$ and $\widehat{N}_t(\Lambda)$. This motivates using a Taylor expansion of the estimator, using the asymptotic form of the well-known delta-method [31] in order to approximate its variance; see [2] for a similar approach for Graph Sample and Hold. The resulting approximation is:

$$\mathrm{Var}(\widehat{N}(\triangle)/\widehat{N}(\Lambda)) \approx \frac{\mathrm{Var}(\widehat{N}(\triangle))}{\widehat{N}(\Lambda)^2} + \frac{\widehat{N}(\triangle)^2 \mathrm{Var}(\widehat{N}(\Lambda))}{\widehat{N}(\Lambda)^4} - 2\frac{\widehat{N}(\triangle)\,\mathrm{Cov}(\widehat{N}(\triangle), \widehat{N}(\Lambda))}{\widehat{N}(\Lambda)^3} \quad (10)$$

Following Theorem 3, $\mathrm{Cov}(\widehat{N}(\triangle), \widehat{N}(\Lambda))$ is estimated as

$$\widehat{V}(\triangle, \Lambda) = \sum_{\substack{\tau \in \widehat{\triangle}, \lambda \in \widehat{\Lambda} \\ \tau \cap \lambda \neq \emptyset}} \widehat{S}_{\tau \cup \lambda} \left(\widehat{S}_{\tau \cap \lambda} - 1 \right) \quad (11)$$

Efficiency. The basic intuition of Algorithm 2 is that the subgraph estimation problem is localized. Hence, all computations can be efficiently performed by exploring the *local neighborhood* of an edge (or a node) [4]. In this section, we discuss how the estimators can be adapted to make the computations more efficient and localized, while still remaining unbiased.

By linearity of expectation, we express $\widehat{N}_t(\triangle)$ (in Corollary 1) as $\widehat{N}_t(\triangle) = 1/3 \sum_{k \in \widehat{K}_t} \widehat{N}_{k,t}(\triangle)$ where $\widehat{N}_{k,t}(\triangle)$ is the conditional estimator of triangle counts for edge k normalized by the number of edges in a triangle. Similarly, we express the unbiased estimator $\widehat{N}_t(\Lambda)$ (in Corollary 2) as $\widehat{N}_t(\Lambda) = 1/2 \sum_{k \in \widehat{K}_t} \widehat{N}_{k,t}(\Lambda)$ where $\widehat{N}_{k,t}(\Lambda)$ is the conditional estimator of wedge counts for edge k normalized by the number of edges in a wedge.

Consider any two distinct edge subsets $J_1, J_2 \subset \widehat{K}_t$. From Theorem 3, the covariance estimator $\widehat{C}_{J_1,J_2,t} = 0$ if J_1 and J_2 are disjoint (*i.e.*, $|J_1 \cap J_2| = 0$). Otherwise, $\widehat{C}_{J_1,J_2,t} > 0$ if and only if their intersection is non-empty (*i.e.*, $|J_1 \cap J_2| > 1$). If J_1, J_2 are triangles (or wedges), then $|J_1 \cap J_2| \leq 1$, since any two *distinct* triangles (or wedges) could never overlap in more than one edge. Thus, the unbiased variance estimators can also be computed locally for each edge, as we show next.

By linearity of expectation, we re-write Equation 8 as follows:

$$\widehat{V}_t(\triangle) = 1/3 \sum_{k \in \widehat{K}_t} \sum_{\tau \in \triangle_t(k)} \widehat{S}_{\tau,t}(\widehat{S}_{\tau,t} - 1) + \sum_{k \in \widehat{K}_t} \sum_{\tau \in \triangle_t(k)} \sum_{\substack{\tau' < \tau \\ \tau' \in \triangle_t(k)}} \widehat{S}_{\tau \setminus \tau',t} \widehat{S}_{\tau' \setminus \tau,t} \widehat{S}_{\tau \cap \tau',t} \left(\widehat{S}_{\tau \cap \tau',t} - 1 \right) \quad (12)$$

Note that for any two distinct triangles $\tau, \tau' \subset \widehat{K}_t$, we have $\widehat{S}_{\tau \cap \tau',t} > 0$ if and only if $\tau \cap \tau' = \{k\}$ for some edge $k \in \widehat{K}_t$. Similarly, we could re-write Equation 9 as follows:

$$\widehat{V}_t(\Lambda) = 1/2 \sum_{k \in \widehat{K}_t} \sum_{\lambda \in \Lambda(k)} \widehat{S}_{\lambda,t}(\widehat{S}_{\lambda,t} - 1) + \sum_{k \in \widehat{K}_t} \sum_{\lambda \in \Lambda_t(k)} \sum_{\substack{\lambda' < \lambda \\ \lambda' \in \Lambda_t(k)}} \widehat{S}_{\lambda \setminus \lambda',t} \widehat{S}_{\lambda' \setminus \lambda,t} \widehat{S}_{\lambda \cap \lambda',t} \left(\widehat{S}_{\lambda \cap \lambda',t} - 1 \right) \quad (13)$$

Algorithm Description. To simplify the presentation of Algorithm 2, we drop the variable t that denote the arrival time in the stream, however the algorithm is valid for any $t \in \mathbb{N}$. We start by calling Algorithm 1 to collect a *weighted sample* $\widehat{K}$ of capacity m edges. For each edge $k \in \widehat{K}$, we use $W(k, \widehat{K}) = 9 * |\widehat{\triangle}(k)| + 1$ where $|\widehat{\triangle}(k)|$ is the number of triangles completed by edge k and whose edges in $\widehat{K}$. Then, we call Algorithm 2 at any time t in the stream to obtain unbiased estimates of triangle/wedge counts, global clustering, and their unbiased variance.

Algorithm 2 Unbiased Estimation of Triangle & Wedge Counts

```
1  procedure GPSESTIMATE(K̂)
2      Initialize all variables to zero
3      parallel for edge k = (v1, v2) ∈ K̂ do
4          q ← min{1, w(k)/z*}
5          for each v3 ∈ Γ̂(v1) do                                  ▷ found wedge
6              k1 ← (v1, v3)
7              q1 ← min{1, w(k1)/z*}
8              /*Compute triangle estimates*/
9              if v3 ∈ Γ̂(v2) then                                  ▷ found triangle
10                 k2 ← (v2, v3)
11                 q2 ← min{1, w(k2)/z*}
12                 N̂k(△) += (qq1q2)^-1                             ▷ triangle count
13                 V̂k(△) += (qq1q2)^-1((qq1q2)^-1 − 1)             ▷ tri. var.
14                 Ĉk(△) += c△ * (q1q2)^-1                         ▷ triangle covariance
15                 c△ = c△ + (q1q2)^-1
16             /*Compute wedge estimates for wedges (v3, v1, v2)*/
17             N̂k(Λ) += (qq1)^-1                                   ▷ wedge count
18             V̂k(Λ) += (qq1)^-1((qq1)^-1 − 1)                     ▷ wedge variance
19             Ĉk(Λ) += cΛ * q1^-1                                 ▷ wedge covariance
20             cΛ = cΛ + q1^-1
21         /*Compute wedge estimates for wedges (v3, v2, v1)*/
22         for each v3 ∈ Γ̂(v2) do
23             k2 ← (v2, v3)
24             q2 ← min{1, w(k2)/z*}
25             N̂k(Λ) += (qq2)^-1                                   ▷ wedge count
26             V̂k(Λ) += (qq2)^-1((qq2)^-1 − 1)                     ▷ wedge variance
27             Ĉk(Λ) += cΛ * q2^-1                                 ▷ wedge covariance
28             cΛ = cΛ + q2^-1
29         Ĉk(△) = Ĉk(△) * 2 * q^-1 * (q^-1 − 1)
30         Ĉk(Λ) = Ĉk(Λ) * 2 * q^-1 * (q^-1 − 1)
31     /*Compute total triangle/wedge estimates*/
32     N̂(△) ← 1/3 * Σ_{k∈K̂} N̂k(△), N̂(Λ) ← 1/2 * Σ_{k∈K̂} N̂k(Λ)
33     V̂(△) ← 1/3 * Σ_{k∈K̂} V̂k(△), V̂(Λ) ← 1/2 * Σ_{k∈K̂} V̂k(Λ)
34     Ĉ(△) ← Σ_{k∈K̂} Ĉk(△), Ĉ(Λ) ← Σ_{k∈K̂} Ĉk(Λ)
35     V̂(△) ← V̂(△) + Ĉ(△)
36     V̂(Λ) ← V̂(Λ) + Ĉ(Λ)
37     return N̂(△), N̂(Λ), V̂(△), V̂(Λ)
```

For each edge $k = (v_1, v_2) \in \widehat{K}$, Alg. 2 searches the neighborhood $\widehat{\Gamma}(v_1)$ of the node with minimum degree (*i.e.*, $\deg(v_1) \leq \deg(v_2)$ for triangles (Line 5). Lines 9–15 compute the estimates for each triangle (k_1, k_2, k) incident to edge k. Lines 17–20 compute the estimates for each wedge (k_1, k) incident to edge k (and centered on node v_1). Lines 25–28 compute the estimates for each wedge (k_2, k) incident to edge k (and centered on node v_2). Finally, the individual edge estimators are summed in Lines 32–36, and returned in Line 37.

We state two key observations in order: First, the estimators of triangle/wedge counts can be computed locally for each sampled edge $k \in \widehat{K}$, while still remaining unbiased. Thus, Algorithm 2 is localized. Second, since the estimators of triangle/wedge counts can be computed for each sampled edge $k \in \widehat{K}$ independently in parallel, Algorithm 2 already has abundant parallelism.

Complexity. Algorithm 2 has a total runtime of $O(m^{3/2})$. This is achieved by $\sum_{(v_1,v_2)\in\widehat{K}} \min\{\deg(v_1), \deg(v_2)\}) = O(a(\widehat{K})m)$, where $a(\widehat{K})$ is the arboricity of the reservoir graph. This complexity can be *tightly bounded* by $O(m^{3/2})$ since $O(a(\widehat{K})m) \leq O(m^{3/2})$ for any sampled graph [9, 4].

Sampling and Counting Cliques and Other Subgraphs. While this section has focused on estimating counts of triangles/wedges and their variances, the statistical framework of Sec. 3 applies to arbitrary subgraphs. We outline the procedure for computing more general subgraph count estimators, specifically for the counts of cliques of some degree d. The disjoining of sampling and estimation steps in our framework allows great flexibility in algorithm design and resource trade-offs. Edge sampling for general cliques can use triangle-count weighting in Alg. 1; this is favorable for general clique count estimation because triangles form sub-cliques of the target d-cliques. Extending the edge sampling weight to include a count of higher-order sub-cliques would increase estimation accuracy at the expense of a higher computational cost. For estimation, d-cliques in the edge sample can be identified using any algorithm designed for this purpose; see e.g., [4]. The count and variance estimates are then computed from the sampling probabilities of the clique edges according to the formulas in Theorems 1, 2 and 3. We stress the flexibility of our framework to target differing classes of target subgraphs, which compares favorably with approaches such as [27] which require the use of subgraph specific data structures.

5. IN-STREAM ESTIMATION

The foregoing analysis enables retrospective subgraph queries: after any number t of stream arrivals have taken place, we can compute an unbiased estimator $\widehat{S}_t(J)$ for any subgraph J. We term this **Post-Stream Estimation**. We now describe a second estimation framework that we call **In-Stream Estimation**. In this paradigm, we can take "snapshots" of specific sampled subgraphs at arbitrary times during the stream, and preserve them as unbiased estimators. These can be used or combined to form desired estimators. These snapshots are not subject to further sampling; their estimates are not updated. However their original subgraphs remain in the graph sample and are subject to sampling in the normal way. Thus we do not change the evolution of the graph sample, we only extract information from it that does not change after extraction. The snapshot times need not be deterministic. For example, each time a subgraph that matches a specified motif appears (e.g. a triangle or other clique) we take a snapshot of the subgraph estimator. If we only need to estimate the number of such subgraphs, it suffices to add the inverse probability of each matching subgraph to a counter.

5.1 Unbiased Estimation with Snapshots

In-stream estimation can be described within the framework of *stopped Martingales* [34]. Return for the moment to our general description in Section 3 of a Martingale $\{X_t : t \in \mathbb{N}\}$ with respect to a filtration $\mathcal{F} = \{\mathcal{F}_t : t \in \mathbb{N}\}$. A random time T is called a **stopping time** w.r.t $\mathcal{F}$ if the event $T \leq t$ is $\mathcal{F}_t$-measurable, i.e., we can decide at time t whether the event has occurred yet. The corresponding **stopped Martingale** is

$$X^T = \{X_t^T : t \in \mathbb{N}\} \text{ where } X_t^T = X_{\min\{T,t\}} \tag{14}$$

Thus, the value of X_t is frozen at T.

We define a **snapshot** as an edge subset $J \subset \mathbb{N}$ and a family $T = \{T_j : j \in J\}$ for $\mathcal{F}_J$-measurable stopping times, giving rise to the product stopped process

$$\widehat{S}_{J,t}^T = \prod_{j\in J} \widehat{S}_{j,t}^{T_j} = \prod_{j\in J} \widehat{S}_{j,\min\{T_j,t\}} \tag{15}$$

Although in-steam estimates use snapshots whose edges have the same stopping time, variance estimation involves products of snapshots with distinct stopping times. Unbiasedness then follows from the following result that applies to any snapshot of the form (15).

THEOREM 4. *(i) $\{\widehat{S}_{J,t}^T : t \geq \max J\}$ is Martingale with respect to $\mathcal{F}_J$ and hence $\mathbb{E}[\widehat{S}_{J,t}^T] = S_{J,t}$.*

(ii) For any set $\mathcal{J}$ of subgraphs of G, each $J \subset \mathcal{J}$ equipped with an $\mathcal{F}_J$-measurable stopping time T_J, then $\sum_{J\in\mathcal{T}_j} \widehat{S}_{J,t}^{T_J}$ is an unbiased estimator of $|\mathcal{J}_t|$.

Algorithm 3 In-Stream Estimation of Triangle & Wedge Counts

```
1  procedure INSTREAM_GPS(K)
2      K̂ ← ∅; z* ← 0
3      while new edge k do
4          GPSESTIMATE(k)
5          GPSUPDATE(k, m)
6      return Ñ(△), Ñ(Λ), Ṽ(△), Ṽ(Λ), Ṽ(△, Λ)
7
8  procedure GPSESTIMATE(k)
9      parallel for Triangle (k1, k2, k) completed by k do
10         if (z* == 0) then q1 ← q2 ← 1
11         else
12             q1 ← min{1, w(k1)/z*}
13             q2 ← min{1, w(k2)/z*}
14         Ñ(△) += 1/(q1 q2)                          ▷ Triangle Count
15         Ṽ(△) += ((q1 q2)^-1 − 1)/(q1 q2)           ▷ Triangle Var.
16         Ṽ(△) += 2(C̃_k1(△) + C̃_k2(△))/(q1 q2)
17         Ṽ(△, Λ) += (C̃_k1(Λ) + C̃_k2(Λ))/(q1 q2)     ▷ Tri.-Wedge Cov.
18         C̃_k1(△) += (q1^-1 − 1)/q2                  ▷ Triangle Covariance
19         C̃_k2(△) += (q2^-1 − 1)/q1
20     parallel for Wedge j ∈ K̂ adjacent to k do
21         if (z* == 0) then q ← 1
22         else q ← min{1, w(j)/z*}
23         Ñ(Λ) += q^-1                               ▷ Wedge Count
24         Ṽ(Λ) += q^-1(q^-1 − 1)                     ▷ Wedge Variance
25         Ṽ(Λ) += 2C̃_j(Λ)/q
26         Ṽ(△, Λ) += C̃_j(△)/q
27         C̃_j(Λ) += 1/q − 1                          ▷ Wedge Covariance
28
29 procedure GPSUPDATE(k, m)
30     Generate u(k) uniformly on (0, 1]
31     w(k) ← W(k, K̂)
32     r(k) ← w(k)/u(k)                               ▷ Priority of edge k
33     K̂ ← K̂ ∪ {k}                                    ▷ Provisionally include edge k
34     C̃_k(△) ← C̃_k(Λ) ← 0
35     if |K̂| > m then
36         k* ← arg min_{k'∈K̂} r(k')                  ▷ Lowest priority edge
37         z* ← max{z*, r(k*)}                        ▷ New threshold
38         K̂ ← K̂ \ {k*}                               ▷ Remove lowest priority edge
39         C̃(△) ← C̃(△) \ C̃_k*(△)                     ▷ Remove covariances of k*
40         C̃(Λ) ← C̃(Λ) \ C̃_k*(Λ)
```

5.2 Variance Estimation for Snapshots

In this section we show how the variance and covariances of snapshots can be estimated as sums of other snapshots involving the stopping times of both constituents. The Martingale formulation is a powerful tool to establish the unbiasedness of the estimates, since the otherwise statistical properties of the product of correlated edge estimators drawn at different times are not evident.

Consider two edge sets J_1 and J_2 each equipped with families of stopping times $T^{(i)} = \{T_j^{(i)} : j \in J_i\}$, with $i = 1, 2$, for the purpose of snapshots. Thus each $j \in J_1 \cap J_2$ is equipped with two generally distinct stopping times $T_j^{(1)}$ and $T_j^{(2)}$, according to its occurrence in the snapshots $\widehat{S}_{J_1,t}^{T^{(1)}}$ and $\widehat{S}_{J_2,t}^{T^{(2)}}$. As an estimator of $\mathrm{Cov}(S_{J_1,t}^{T^{(1)}}, S_{J_2,t}^{T^{(2)}})$ we will use:

$$\widehat{C}_{J_1,J_2,t}^{T^{(1)},T^{(2)}} = \widehat{S}_{J_1,t}^{T^{(1)}} \widehat{S}_{J_2,t}^{T^{(2)}} - \widehat{S}_{J_1\setminus J_2,t}^{T^{(1)}} \widehat{S}_{J_2\setminus J_1,t}^{T^{(2)}} \widehat{S}_{J_1\cap J_2,t}^{T^{(1)}\vee T^{(2)}} \quad (16)$$

where $T^{(1)} \vee T^{(2)} = \{\max\{T_j^{(1)}, T_j^{(2)}\} : j \in J_1 \cap J_2\}$, i.e., we use the earlier stopping time for edges common to both subsets.

THEOREM 5. *(i)* $\widehat{C}_{J_1,J_2,t}^{T^{(1)},T^{(2)}}$ *is an unbiased estimator of* $\mathrm{Cov}(\widehat{S}_{J_1,t}^{T^{(1)}}, \widehat{S}_{J_2,t}^{T^{(2)}})$.

(ii) $\widehat{C}_{J_1,J_2,t}^{T^{(1)},T^{(2)}} \geq 0$ *and hence* $\mathrm{Cov}(\widehat{S}_{J_1,t}^{T^{(1)}}, \widehat{S}_{J_2,t}^{T^{(2)}}) \geq 0$.

(iii) $\widehat{S}_{J,t}^{T}(\widehat{S}_{J,t}^{T} - 1)$ *is an unbiased estimator of* $\mathrm{Var}(\widehat{S}_{J,t}^{T})$.

(iv) $\widehat{C}_{J_1,J_2,t}^{T^{(1)},T^{(2)}} = 0$ *if and only if* $\widehat{S}_{J_1,t}^{T^{(1)}} = 0$ *or* $\widehat{S}_{J_2,t}^{T^{(2)}} = 0$, *i.e., covariance estimators are computed only from snapshots that have been sampled.*

Covariance Estimation for Post-Stream Estimation. Post-stream estimation is a special case of in-stream estimation with all $T_j = \infty$. We recover the corresponding Thm 3 concerning covariances from Thm 5 by omitting all stopping times T_j from the notation.

5.3 In-Stream Triangle & Wedge Counts

We now describe an application of In-Stream Estimation to Triangle Sampling and Counting. We sample from the stream based on the previous triangle count weighting, but the triangle counting is performed entirely in-stream. The full process is described in Algorithm 3. In this section we state and prove the form of estimators for triangle count and its variance, and describe the corresponding steps in the algorithm. Space limitations preclude giving a similar level of detail for wedge counting and edge-wedge covariance.

Unbiased In-Stream Estimation of Triangle Count. Using our previous notation $\triangle_t$ to denote the set of triangles all of whose edges have arrived by t, we write each such triangle as (k_1, k_2, k_3) with k_3 the last edge to arrive. If k_1 and k_2 are in the sample $\widehat{K}$ when k_3 arrives, we take a snapshot of the wedge (k_1, k_2) *prior to the sampling step for* k_3. Formally, we let T_{k_3} denote the slot immediately prior to arrival of k_3 and form the snapshot $\widehat{S}_{\{k_1,k_2\}}^{T_{k_3}}$. T_{k_3} is a stopping time because edge arrivals are deterministic.

THEOREM 6. $\widetilde{N}_t(\triangle) = \sum_{(k_1,k_2,k_3)\in\triangle_t} \widehat{S}_{\{k_1,k_2\},t}^{T_{k_3}}$ *is an unbiased estimator of* $N_t(\triangle)$.

Estimation of Triangle Count Variance. We add some further notation in preparation for estimating the variance of $\widetilde{N}_t(\triangle)$. Let $\widehat{K}_{[t]} = \cup_{t>0}\widehat{K}_t$ denote the set of edges that are sampled at any time up to t. Let $\widehat{\triangle}_t = \{(k_1, k_2, k_3) \in \triangle_t : \widehat{S}_{\{k_1,k_2\}}^{T_{k_3}} > 0$ denote the (random) subset of all triangles in $\triangle_t$ that have positive snapshot. Let $\widehat{K}_{[t]}^{(2)}(k)$ denote the set of pairs (j', k') of edges in $\widehat{K}_{[t]}$ such that each of the edge pairs (k, j') and (k, k') are the first two edges in distinct triangles in $\widehat{\triangle}_t$, and with (j', k') ordered by their stopping time of the third edges in these triangles, i.e., $T_{k,j'} < T_{k,k'}$.

THEOREM 7. $\mathrm{Var}(\widetilde{N}_t(\triangle))$ *has unbiased estimator*

$$\widetilde{V}_t(\triangle) = \sum_{(k_1,k_2,k_3)\in\widehat{\triangle}_t} \frac{1}{p_{k_1,T_{k_3}} p_{k_2,T_{k_3}}} \left(\frac{1}{p_{k_1,T_{k_3}} p_{k_2,T_{k_3}}} - 1\right) \quad (17)$$

$$+2 \sum_{k\in\widehat{K}_{[t]}} \sum_{\substack{(j',k')\in \\ \widehat{K}_{[t]}^{(2)}(k)}} \frac{1}{p_{k',T_{k,k'}} p_{j',T_{k,j'}} p_{k,T_{k,k'}}} \left(\frac{1}{p_{k,T_{k,j'}}} - 1\right)$$

Description of Algorithm 3. We now describe the portions of Algorithm 3 relating the in-stream estimator $\widetilde{N}(\triangle)$ and $\widetilde{V}(\triangle)$. When an edge k arrives, an update is performed for each triangle that k completes (line 9). These updates can be performed in parallel because each such triangle must have distinct edges other than k. Triangle count $\widetilde{N}(\triangle)$ is updated with the current inverse probabilities of its first two edges k_1 and k_2 (line 14). The variance $\widetilde{V}(\triangle)$ is updated first with the variance term for the current triangle (line 15) then secondly with its cumulative contributions $\widehat{C}_{k_1}(\triangle)$ and

$\widehat{C}_{k_1}(\triangle)$ to the covariances with all previous triangles whose first two edges include k_1 or k_2 (line 16). These cumulative terms are then updated by the current triangle (lines 18 and 19). Wedge count variables are updated in a similar manner in lines 20–27. The edge-wedge covariance $\widetilde{V}(\triangle, \Lambda)$ used for estimation of the global clustering coefficient α is updated using the cumulative triangle and wedge covariances in lines 17 and 26.

6. EXPERIMENTS & RESULTS

We test the performance of graph priority sampling on a large set of graphs up to billions of nodes/edges, selected from a variety of domains and types, such as social, web, among others. All graph data sets are available for download [29] [1]. For all graph datasets, we consider an undirected, unweighted, simplified graph without self loops. We generate the graph stream by randomly permuting the set of edges in each graph. For each graph, we perform ten different experiments with sample sizes in the range of 10K–1M edges. We test GPS as well as baseline methods using a single pass over the edge stream (such that each edge is processed once): both GPS post and in-stream estimation randomly select the same set of edges with the same random seeds. Thus, the two methods only differ in the estimation procedure. For these experiments, we used a server with two Intel Xeon E5-2687W 3.1GHz CPUs. Each processor has 8 cores with 20MB of L3 cache and 256KB of L2 cache, and 128GB of memory. The experimental setup is executed independently for each sample size as follows:

1. For each sample size m, Alg 1 collects an edge sample $\widehat{K} \subset K$.
2. We use Alg 2 for post stream estimation, where the estimates are obtained only from the sample. We use Alg 3 for in-stream estimation, where the estimates are updated incrementally in a single pass during the sampling process. Thus, both GPS post and in-stream estimation use the same sample.
3. Given a sample $\widehat{K} \subset K$, we use the absolute relative error (ARE) $|\mathbb{E}[\widehat{X}] - X|/X$ to measure the deviation between the expected value of the estimates $\widehat{X}$ and the actual statistics X. We use X to refer to triangle, wedge counts, or global clustering.
4. We compute 95% confidence bounds as $\widehat{X} \pm 1.96\sqrt{\text{Var}[\widehat{X}]}$ [32].

Error Analysis and Confidence Bounds. Table 2 summarizes the main graph properties and compares GPS post stream and in-stream estimation for a variety of graphs at sample size $m = 200$K edges. First, we observe that GPS in-stream estimation has on average $< 1\%$ relative error across most graphs. In addition, GPS post stream estimation has on average $\leq 2\%$. Thus, both methods provide accurate estimates for large graphs with a small sample size. Table 2 also shows that the upper and lower bounds of GPS in-stream estimation are smaller than those obtained using GPS post stream estimation. Both methods use the same sample. However, a key advantage for GPS in-stream estimation versus GPS post stream estimation is its ability to minimize the variance of the estimators. Thus, GPS in-stream estimates are not only accurate but also have a small variance and tight confidence bounds.

Second, we observe that the GPS framework provides high quality general samples to accurately estimate various properties simultaneously. For example, Table 2 shows consistent performance across all graphs for the estimation of triangle/wedge counts and global clustering with the same sample. Similarly, in Figure 1, we observe that GPS accurately estimates both triangle and wedge counts simultaneously with a single sample, with a relative error of 0.6% for for both triangle and wedge counts.

[1] www.networkrepository.com

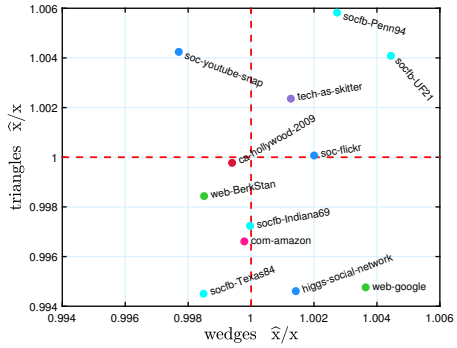

Figure 1: Comparing $\hat{x}/x$ of triangles and wedges. The closer the points are to the intersection of the red lines (actual) the better. Points are colored by graph type. Results are from the in-stream estimation method at 100K.

Table 1: Estimates of expected value and relative error using GPS in-stream with 1M edges for a representative set of billion-scale graphs. The number of triangles in the full graph is shown in the fourth column. syn$(|V|, \rho)$ is an Erdős-Rényi graph with $|V|$ is the number of nodes and ρ is the density.

graph	$\lvert K\rvert$	$\frac{\lvert\widehat{K}\rvert}{\lvert K\rvert}$	X	$\widehat{X}$	$\frac{\lvert X-\widehat{X}\rvert}{X}$	LB	UB
soc-friendster	1.8B	0.0006	4.17B	4.17B	**0.0004**	4.02B	4.3B
web-Clueweb09	7.9B	0.0001	31B	31B	**0.009**	29.9B	32.7B
syn$(100\text{M}, 10^{-6})$	4.9B	0.0002	165.8K	166K	**0.001**	135.2K	196.9K
syn$(100\text{M}, 10^{-5})$	49.9B	0.00002	165.4M	165.1M	**0.001**	155.3M	174.8M

Finally, we investigate the properties of the sampling distribution and the convergence of the estimates as the sample size increases between 10K–1M edges (See Figure 2). We used graphs from various domains and types. We observe that the confidence intervals of triangle counts are small in the range 0.90–1.10 for most graphs at 40K sample size. Notably, for a large Twitter graph with more than 260M edges (soc-twitter-2010), GPS in-stream estimation accurately estimates the triangle count with $< 1\%$ error, while storing 40K edges, which is only a fraction of 0.00015 of the total number of edges in the graph. Due to space limitations, we removed the confidence plots for wedges and clustering coefficient. However, we observe that the confidence interval are very small in the range of 0.98–1.02 for wedges, and 0.90–1.10 for global clustering.

Results for both massive real-world and synthetic graphs of up to 49B edges are provided in Table 1. Synthetic graphs were generated using Erdős-Rényi graph model with different densities. Notably, GPS is shown to be accurate with less than 0.01 error. We observed similar results for wedges and clustering coefficient, the details were removed due to space constraints.

Baseline Study. The state-of-the-art algorithms for triangle count estimation in adjacency graph streams are due to the neighborhood sampling (NSAMP) in [27] and the triangle sampling (TRIEST) in [14]. We discuss their performance in turn compared with GPS post stream estimation. We also compare with MASCOT [24] and simple uniform random sampling (SRS) [33]. Table 3 summarizes the results of the comparison. Our implementation of the NSAMP [27] algorithm follows the description in the paper, which achieves a near-linear total time if and only if running in bulk-processing. Otherwise the algorithm is too slow and not practical even for medium size graphs with a total time of $\mathcal{O}(|K|r)$. Overall, GPS post stream estimation achieves 98%–99% accuracy, while NSAMP achieves only 80%–84% accuracy for most graphs and 92% accuracy for higgs-soc-net graph. Our implementation of

Table 2: Estimates of expected value and relative error using 200K edges for a representative set of 10 graphs. The graphlet statistic for the full graph is shown in the fourth column. LB and UB are 95% confidence lower and upper bounds, respectively.

				ACTUAL	GPS IN-STREAM				GPS POST STREAM			
	graph	$\|K\|$	$\frac{\|\widehat{K}\|}{\|K\|}$	X	$\widehat{X}$	$\frac{\|X-\widehat{X}\|}{X}$	LB	UB	$\widehat{X}$	$\frac{\|X-\widehat{X}\|}{X}$	LB	UB
TRIANGLES	ca-hollywood-2009	56.3M	0.0036	4.9B	4.9B	**0.0009**	4.8B	5B	4.8B	**0.0036**	4.6B	5.1B
	com-amazon	925.8K	0.216	667.1K	667.2K	**0.0001**	658.5K	675.8K	666.8K	**0.0004**	653.6K	680K
	higgs-social-network	12.5M	0.016	83M	82.6M	**0.0043**	80.8M	84.4M	83.2M	**0.0031**	79.5M	87M
	soc-livejournal	27.9M	0.0072	83.5M	83.1M	**0.0043**	80.6M	85.7M	81.5M	**0.0244**	72M	91M
	soc-orkut	117.1M	0.0017	627.5M	625.8M	**0.0028**	601.4M	650.1M	614.8M	**0.0203**	396M	833.7M
	soc-twitter-2010	265M	0.0008	17.2B	17.3B	**0.0009**	16.8B	17.7B	17.3B	**0.0027**	13.3B	21.3B
	soc-youtube-snap	2.9M	0.0669	3M	3M	**0.0004**	2.9M	3.1M	3M	**0.0003**	2.9M	3.1M
	socfb-Texas84	1.5M	0.1257	11.1M	11.1M	**0.0011**	10.9M	11.3M	11.1M	**0.0013**	10.4M	11.9M
	tech-as-skitter	11M	0.018	28.7M	28.5M	**0.0081**	27.7M	29.3M	28.3M	**0.0141**	26.5M	30.1M
	web-google	4.3M	0.0463	13.3M	13.4M	**0.0034**	13.2M	13.6M	13.4M	**0.0078**	13.1M	13.8M
WEDGES	ca-hollywood-2009	56.3M	0.0036	47.6B	47.5B	**0.0011**	47.3B	47.8B	47.5B	**0.0026**	46.9B	48.1B
	com-amazon	925.8K	0.216	9.7M	9.7M	**0.0002**	9.7M	9.8M	9.7M	**0.0021**	9.6M	9.9M
	higgs-social-network	12.5M	0.016	28.7B	28.7B	**0.001**	28.5B	28.9B	28.7B	**0.0008**	28.1B	29.3B
	soc-livejournal	27.9M	0.0072	1.7B	1.7B	**0.0005**	1.7B	1.8B	1.8B	**0.0008**	1.7B	1.8B
	soc-orkut	117.1M	0.0017	45.6B	45.5B	**0.0016**	45B	46B	45.5B	**0.0009**	44.3B	46.8B
	soc-twitter-2010	265M	0.0008	1.8T	1.8T	**0.0002**	1.8T	1.8T	1.8T	**0.0016**	1.7T	1.8T
	soc-youtube-snap	2.9M	0.0669	1.4B	1.4B	**0.0035**	1.4B	1.4B	1.4B	**0.0084**	1.4B	1.5B
	socfb-Texas84	1.5M	0.1257	335.7M	334.9M	**0.0022**	331.4M	338.5M	335.1M	**0.0017**	323M	347.2M
	tech-as-skitter	11M	0.018	16B	16B	**0.0005**	15.8B	16.1B	15.9B	**0.0016**	15.6B	16.3B
	web-google	4.3M	0.0463	727.4M	728.8M	**0.002**	721M	736.7M	732.2M	**0.0066**	711.8M	752.5M
CLUSTERING COEFF. (CC)	ca-hollywood-2009	56.3M	0.0036	0.31	0.31	**0.002**	0.306	0.315	0.309	**0.0009**	0.295	0.323
	com-amazon	925.8K	0.216	0.205	0.205	$<$**10**$^{-4}$	0.203	0.208	0.205	**0.0025**	0.201	0.209
	higgs-social-network	12.5M	0.016	0.009	0.009	**0.0034**	0.008	0.009	0.009	**0.0039**	0.008	0.009
	soc-livejournal	27.9M	0.0072	0.139	0.139	**0.0039**	0.135	0.143	0.136	**0.0252**	0.12	0.151
	soc-orkut	117.1M	0.0017	0.041	0.041	**0.0012**	0.04	0.043	0.04	**0.0193**	0.026	0.055
	soc-twitter-2010	265M	0.0008	0.028	0.028	**0.0012**	0.028	0.029	0.028	**0.0004**	0.022	0.035
	soc-youtube-snap	2.9M	0.0669	0.006	0.006	**0.0032**	0.006	0.006	0.006	**0.0088**	0.006	0.007
	socfb-Texas84	1.5M	0.1257	0.1	0.1	**0.0012**	0.098	0.102	0.1	**0.0031**	0.093	0.107
	tech-as-skitter	11M	0.018	0.005	0.005	**0.0076**	0.005	0.006	0.005	**0.0124**	0.005	0.006
	web-google	4.3M	0.0463	0.055	0.055	**0.0014**	0.054	0.056	0.055	**0.0013**	0.053	0.057

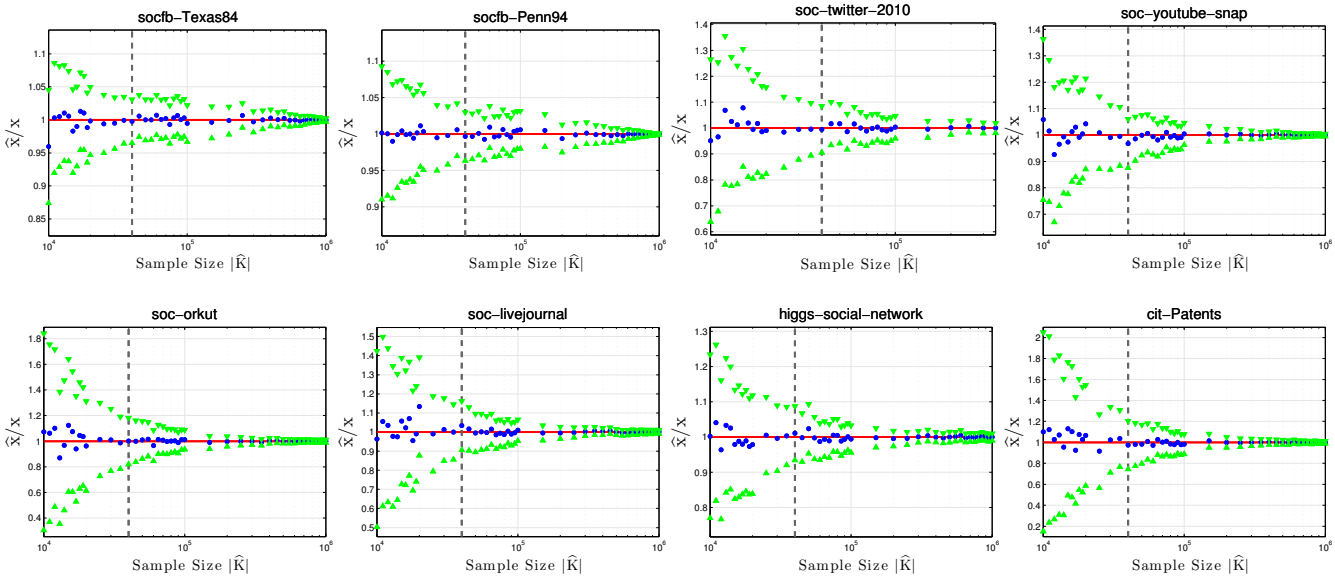

Figure 2: Confidence bounds for Graph Priority Sampling with instream estimation of triangle counts. We used graphs from a variety of domains and types. The properties of the sampling distribution and convergence of the estimates are investigated as the sample size increases. The circle (●) represents $\widehat{X}/X$ (y-axis) whereas ▲ and ▼ are LB/X and UB/X, respectively. Dashed vertical line (grey) refers to the sample at 40K edges. Notably, the proposed framework has excellent accuracy even at this small sample size.

the TRIEST [14] algorithm follows the main approach in the paper. TRIEST was unable to produce a reasonable estimate showing only 60%–82% accuracy. Similarly, MASCOT achieves only 35%–79% accuracy and simple random sampling achieves only 19%–66% accuracy. Thus, GPS post stream estimation outperforms the four baseline methods. Table 3 also shows the average update time per edge (in microseconds). We note that GPS post stream estimation achieves an average update time that is 35x–56x faster than NSAMP with bulk-processing (for cit-Patents and infra-roadNet-CA graphs and at least 2x faster for higgs-soc-net).

TRIEST, MASCOT, and SRS use an average of 3, 2, and 2.5 microseconds/edge respectively. In Table 3, we observe the update time for higgs-soc-net to be slower compared to the other graphs. This is likely due to higgs-soc-net having a much larger local clustering than the other graphs which in turn triggers more updates in the GPS sample. However, the error from GPS remains small and is orders of magnitude smaller than the state-of-the-art methods.

Table 3: Baseline comparison at sample size ≈100K

	NSAMP	TRIEST	MASCOT	SRS	GPS POST
	Absolute Relative Error (ARE)				
cit-Patents	0.192	0.401	0.65	0.81	0.008
higgs-soc-net	0.079	0.174	0.209	0.34	0.011
infra-roadNet-CA	0.165	0.301	0.39	0.48	0.013
	Average Time (μ_s/edge)				
cit-Patents	34.2	3.01	2.02	2.5	0.63
higgs-soc-net	26.08	4.40	2.02	2.2	11.74
infra-roadNet-CA	28.72	2.81	2.05	2.5	0.831

We also compared to other methods in [20] and [8] (results omitted for brevity). Even though the method in [8] is fast, it fails to find a triangle most of the time, producing low quality estimates (mostly zero estimates). On the other hand, the method of [20] is too slow for extensive experiments with $\mathcal{O}(m)$ update complexity per edge (where m is the reservoir size). GPS post stream estimation achieves at least 10x accuracy improvement compared to their method.

Unbiased Estimation vs. Time. We now track the estimates as the graph stream progresses one edge at a time, starting from an empty graph. Figure 3 shows GPS estimates for triangle counts and clustering coefficient as the stream is evolving overtime. Notably, the estimates are indistinguishable from the actual values. Figure 3 also shows the 95% confidence upper and lower bounds. These results are for a sample of 80K edges (a small fraction $\leq 1\%$ of the size of soc-orkut) using GPS in-stream.

Table 4: Mean absolute relative error for estimates of triangle counts vs. time (sample size = 80K).

graph	Algorithm	Max. ARE	MARE
ca-hollywood-2009	TRIEST	0.492	0.211
	TRIEST-IMPR	0.066	0.018
	GPS POST	0.049	0.020
	GPS IN-STREAM	0.016	0.003
tech-as-skitter	TRIEST	0.628	0.249
	TRIEST-IMPR	0.134	0.048
	GPS POST	0.087	0.035
	GPS IN-STREAM	0.032	0.014
infra-roadNet-CA	TRIEST	0.98	0.47
	TRIEST-IMPR	0.33	0.09
	GPS POST	0.15	0.05
	GPS IN-STREAM	0.058	0.02
soc-youtube-snap	TRIEST	0.362	0.119
	TRIEST-IMPR	0.049	0.016
	GPS POST	0.022	0.009
	GPS IN-STREAM	0.020	0.008

Given the slow execution of NSAMP, we compare against TRIEST and its improved estimation procedure (TRIEST-IMPR). Note that TRIEST and TRIEST-IMPR are both using the same sample and random seeds. The two approaches are based on reservoir sampling [33]. We used graphs from a variety of domains and types for this comparison, and all methods are using the same sample size. We measure the error using the Mean Absolute Relative Error (MARE) $\frac{1}{T}\sum_{t=1}^{T}\frac{|\widehat{X}_t - X_t|}{X_t}$, where T is the number of time-stamps. We also report the maximum error $\max_{t=1}^{T}\frac{|\widehat{X}_t - X_t|}{X_t}$.

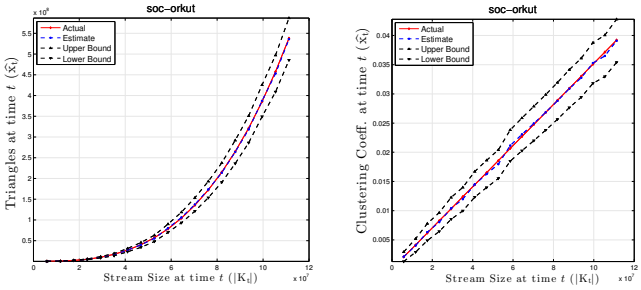

Figure 3: Graph Priority Sampling with in-stream estimation versus time. Results for social and tech networks at sample size 80K edges for triangle counts and global clustering with 95% confidence lower and upper bounds.

Table 4 summarizes the comparison results. For all graphs, GPS with in-stream estimation outperforms both TRIEST and its improved estimation procedure TRIEST-IMPR. We note that indeed TRIEST-IMPR significantly improves the quality of the estimation of TRIEST. However, in comparison with GPS we observe that GPS with post stream estimation is orders of magnitude better than TRIEST, which shows that the quality of the sample collected by GPS is much better than TRIEST (regardless the estimation procedure used, whether it is post or in-stream estimation).

Impact of Stream Ordering on Error and Update Time. Results for different ordering strategies are shown in Table 5. We consider both the arbitrary stream order where the graph is presented as a sequence of edges in any arbitrary order and there is no bound on the degree of a vertex, and the incidence stream order where all edges incident to a vertex are presented successively. We observe accurate results with no significant difference in error between the ordering schemes. Note that arbitrary ordered streams is much harder since edges that participate in triangles would appear at random times. Nevertheless, the proposed approach works well for different ordering strategies as shown in Table 5. One real-world example of incidence ordering pertains to streams with bursty behavior (*e.g.*, where all edges adjacent to a node appear in order). For example, on Twitter when a celebrity/influencer tweets, and then is retweeted immediately after by many users/followers. In terms of the average update time, we observe the incidence ordering to be faster than arbitrary which is likely due to caching and locality benefits.

Table 5: Impact of the ordering of the stream on error and update time. Estimates are from GPS in-stream with 1M edges. Recall that ARE is the Absolute Relative Error.

Graph	Stream order	$\frac{\lvert\widehat{K}\rvert}{\lvert K\rvert}$	ARE	Avg. time (μ_s/edge)	LB/X	UB/X
soc-friendster	ARBITRARY	0.0006	0.003	2.65	0.95	1.05
	INCIDENCE	0.0006	0.0004	0.53	0.96	1.037
web-ClueWeb50m	ARBITRARY	0.002	0.004	3.66	0.97	1.018
	INCIDENCE	0.002	0.0006	1.92	0.984	1.016

Impact of Sampling Weight on Error and Variance. When a new edge k arrives in the stream, Alg. 1 computes a weight function $W(k, \widehat{K})$. We consider three different strategies for sampling weights: (1) triangle-based weights which use the number of completed triangles by k whose other edges in $\widehat{K}$ (used in all the previous results, see Sec. 4 for details), (2) wedge-based weights which use the number of completed wedges by k whose other edges in $\widehat{K}$, and (3) uniform weights for all incoming edges (this is equivalent to simple random sampling [33]). We observed that triangle-based

weights is the most accurate with less variance among the three weighting schemes for the estimation of triangle counts. We observed significant difference between weighting schemes for GPS post stream estimation. For soc-friendster graph with $m = 1$M edges, we obtained 25% error using wedge-based weights, 43% error using uniform weights, and 3% error using triangle-based weights. The estimator variance was 3.8x higher using wedge-based weights, and 6.2x higher using uniform weights compared to triangle-based weights.

7. RELATED WORK

Sampling from graph streams. There has been great interest in algorithms for mining massive and dynamic graphs from data streams, many based on sampling and approximation techniques. An early such work [19] concerned following paths and connectivity in directed graphs. Much earlier work on graph streams focused on the semi-streaming model [26, 18], where the algorithm is allowed to use $\mathcal{O}(n \operatorname{polylog} n)$ space to solve graph problems that are provably intractable in sub-linear space. Recent work focused on graph mining problems such as finding common neighbors [7], estimation of pagerank [30], clustering and outlier detection [1], multigraph streams [13], link prediction [35], among others [3].

Sampling and Estimation of Subgraphs. Subgraph counting (in particular triangle counting) has gained significant attention due to its applications in social, information, and web networks. In static graphs that fit in memory, recent methods focused on estimation of local and global counts of motifs of size 4 nodes, *e.g.* [5]. In streaming graphs, most work focused on estimating only the number of triangles. Early work [8] provides a space-bounded algorithm for the estimation of triangle counts and clustering coefficient in the incidence graph stream model. However, these guarantees no longer hold for the adjacency stream model, where the edges arrive in arbitrary order. A single pass streaming algorithm incurring $\mathcal{O}(m\Delta/T)$-space, where Δ is the maximum degree is proposed in [27]. However, this algorithm requires both large storage and update overhead to provide accurate results. For example, their algorithm needs at least 128 estimators (*i.e.*, storing more than 128K edges) and uses large batch sizes (*e.g.*, a million edges) to obtain accurate/efficient results. A single-pass $\mathcal{O}(m/\sqrt{T})$-space streaming algorithm was proposed in [20] for transitivity estimation with arbitrary additive error guarantees. It maintains two reservoirs, the first to select a uniform sample of edges from the stream, and the second to select a uniform sample of wedges from the first.

Other approaches focused on maintaining a set of edges sampled randomly from the graph stream. *Graph sample-and-hold* [2] is a framework for unbiased estimation of subgraph counts, however, it does not provide a fixed-size sample. A similar approach was recently proposed for local (node/edge) triangle count estimation in graph streams [24]. Other methods extend reservoir sampling to graph streams. For example, reservoir sampling has been used for detecting outliers in graph streams [1], estimating the distribution of various graph properties (*e.g.*, path length, clustering) [3], and estimating triangle counts in dynamic graph streams with insertions and deletions [14].

8. CONCLUSION

We have presented *graph priority sampling*, a new framework for order-based reservoir sampling from massive graph streams. GPS provides a general way to weight edge sampling according to auxiliary variables to estimate various graph properties. We showed how edge sampling weights can be chosen so as to minimize the estimation variance of counts of subgraphs, such as triangles and wedges. Unlike previous graph sampling algorithms, GPS differentiates between the functions of sampling and estimation. We proposed two estimation approaches: (1) Post-stream estimation, to allow GPS to construct a reference sample of edges to support retrospective graph queries, and (2) In-stream estimation, to allow GPS to obtain lower variance estimates by incrementally updating the count estimates during stream processing. We provided a novel Martingale formulation for subgraph count estimation. We performed a large-scale experimental analysis. The results show that GPS achieves high accuracy with $< 1\%$ error on large real-world graphs from a variety of domains and types, while storing a small fraction of the graph and average update times of a few microseconds per edge.

9. PROOFS OF THEOREMS

LEMMA 5. *For each t, the events* $\{\{j \in K_t\} : j \le t\}$ *are measurable w.r.t* $\mathcal{F}_{i,t}$.

PROOF OF LEMMA 5. The proof is by induction. It is trivially true fot $t = i$. Assume true for $t-1$, then membership of K_{t-1} is $\mathcal{F}_{i,t-1}$ measurable, and hence so is membership of K'_{t-1}. Selection of i is clearly $\mathcal{F}_{i,t}$-measurable, and if i is selected, the remaining selections are $\mathcal{F}^{(0)}_{i,t}$-measurable since then $z_{\{ij\},t} = z_{j,t}$. □

PROOF OF THEOREM 1. Lemma 5 established measurability. For $t \ge i$, since $R_{i,t}$ is $\mathcal{Z}_{i,t}$-measurable, conditioning first on $z_{i,t}$:

$$\mathbb{E}[\widehat{S}_{i,t}|z_{i,t},\mathcal{F}_{i,t-1}] = \frac{1}{R_{i,t}}\mathbb{E}[I(B_i(z^*_{i,t}))|z_{i,t},\mathcal{F}_{i,t-1}] \quad (18)$$

Since $B_i(z^*_{i,t}) = B_i(z_{i,t}) \cap B_i(z^*_{i,t-1})$ and $I(B_i(z^*_{t-1}))$ is $\mathcal{F}_{i,t-1}$-measurable, then for any event on which the conditional expectation (18) is positive, we have

$$\begin{aligned}\mathbb{E}[I(B_i(z^*_{i,t}))|z_{i,t},\mathcal{F}_{i,t-1}] &= \mathbb{P}[B_i(z_{i,t})|B_i(z^*_{i,t-1}),\mathcal{Z}_{i,t},\mathcal{F}^{(0)}_{i,t-1}]\\ &= \mathbb{P}[B_i(z^*_{i,t})|\mathcal{Z}_{i,t},w_i]/\mathbb{P}[B_i(z^*_{i,t-1})|\mathcal{Z}_{i,t},w_i]\\ &= \mathbb{P}[w_i/u_i > z^*_{i,t}|\mathcal{Z}_{i,t},w_i]/\mathbb{P}[w_i/u_i > z^*_{i,t-1}|\mathcal{Z}_{i,t},w_i]\\ &= R_{i,t}/R_{i,t-1} \quad (19)\end{aligned}$$

since once we have conditioned on $B_i(z^*_{i,t-1})$ and $\mathcal{Z}_{i,t}$, then $\mathcal{F}^{(0)}_{i,t-1}$ conditions only through the dependence of w_i on the sample set K_{i-1}. Thus we have established that $\mathbb{E}[\widehat{S}_{i,t}|z_{i,t},\mathcal{F}_{i,t-1}] = \widehat{S}_{i,t-1}$ regardless of $z_{i,t}$, and hence $\mathbb{E}[\widehat{S}_{i,t}|\mathcal{F}_{i,t-1}] = \widehat{S}_{i,t-1}$. □

PROOF OF THEOREM 4. (ii) follows from (i) by linearity of expectation. For (i), observe that $\widehat{S}^{T_j}_{j,t} = S^{T_j}_{j,t-1} + I(T \ge t)(\widehat{S}_{j,t} - \widehat{S}_{j,t-1})$; one checks that this reproduces $\widehat{S}_{j,\min\{t,T_j\}}$. Thus

$$\widehat{S}^{T}_{J,t} = \prod_{j\in J}\widehat{S}^{T_j}_{j,t-1} + \sum_{L\subsetneq J}\prod_{\ell\in L}S^{T_\ell}_{j,\ell-1}\prod_{j\in J\setminus L}I(T_j \ge t)(\widehat{S}_{j,t} - \widehat{S}_{j,t-1})$$

Observe that $I(T_j \ge t) = 1 - I(T_j \le t-1)$ is in fact $\mathcal{F}_{J,t-1}$-measurable. Hence taking expectations w.r.t. $\mathcal{F}_{J,t-1}$ then the product form from Theorem 2 tells us that for any $L \subsetneq J$

$$\begin{aligned}&\mathbb{E}[\prod_{j\in J\setminus L} I(T_j \ge t)(\widehat{S}_{j,t} - \widehat{S}_{j,t-1})|\mathcal{F}_{J,t-1}]\\ &\quad = \prod_{j\in J\setminus L} I(T_j \ge t)\mathbb{E}[\widehat{S}_{j,t} - \widehat{S}_{j,t-1}|\mathcal{F}_{J,t-1}] = 0 \quad (20)\end{aligned}$$

and hence $\mathbb{E}[\widehat{S}^T_{J,t}|\mathcal{F}_{J,t-1}] = \prod_{j\in J}\widehat{S}^{T_j}_{j,t-1} = \widehat{S}^T_{J,t-1}$ □

PROOF OF THEOREM 5. (i)

$$\begin{aligned}\operatorname{Cov}(\widehat{S}^{T^{(1)}}_{J_1,t}, \widehat{S}^{T^{(2)}}_{J_2,t}) &= \mathbb{E}[\widehat{S}^{T^{(1)}}_{J_1,t}\widehat{S}^{T^{(2)}}_{J_2,t}] - \mathbb{E}[\widehat{S}^{T^{(1)}}_{J_1,t}]\mathbb{E}[\widehat{S}^{T^{(2)}}_{J_2,t}]\\ &= \mathbb{E}[\widehat{S}^{T^{(1)}}_{J_1,t}S^{T^{(2)}}_{J_2,t}] - 1 \quad (21)\end{aligned}$$

Hence the results follows because

$$E[\widehat{S}^{T^{(1)}}_{J_1\setminus J_2,t}\widehat{S}^{T^{(2)}}_{J_2\setminus J_1,t}\widehat{S}^{T^{(1)}\vee T^{(2)}}_{J_1\cap J_2,t}] = 1 \quad (22)$$

from Theorem 4 since $J_1 \setminus J_2$, $J_2 \setminus J_1$ and $J_1 \cap J_2$ are disjoint.

(ii)$\widehat{C}^{T^{(1)},T^{(2)}}_{J_1,J_2,t} = \widehat{S}^{T^{(1)}}_{J_1\setminus J_2,t}\widehat{S}^{T^{(2)}}_{J_2\setminus J_1,t}(\widehat{S}^{T^{(1)}}_{J_1\cap J_2}\widehat{S}^{T^{(2)}}_{J_1\cap J_2} - \widehat{S}^{T^{(1)}\vee T^{(2)}}_{J_1\cap J_2,t})$. which is nonnegative since each $j \in J_1 \cap J_2$ brings a factor of the form $1/(p^{(1)}p^{(2)})$ to $\widehat{S}^{T^{(1)}}_{J_1\cap J_2}\widehat{S}^{T^{(2)}}_{J_1\cap J_2}$, where $p^{(i)} = p_{j,\max\{t,T^{(i)}_j\}}$. This exceeds the matching term in $\widehat{S}^{T^{(1)}\vee T^{(2)}}_{J_1\cap J_2,t}$, i.e., $1/\min\{p_1,p_2\}$.

(iii) Follows from (i) upon setting $J_1 = J_2 = J$.

(iv) The equivalence clearly applies to the first monomial in (16). For the second monomial, note that $(\widehat{S}^{T^{(1)}}_{J_1,t} = 0) \wedge (\widehat{S}^{T^{(2)}}_{J_2,t} = 0)$ if and only if $\widehat{S}^{T^{(1)}}_{j,t} = 0$ for some $j \in J_1$ or $\widehat{S}^{T^{(2)}}_{j,t} = 0$ for some $j \in J_2$. If this condition holds for some $j \in J_1 \Delta J_2$ we are done. Otherwise, we require $\widehat{S}^{T^{(i)}_j}_{j,t} = 0$ for some $j \in J_1 \cap J_2$ and some $i \in \{1,2\}$. But for $j \in J_1 \cap J_2$, $\widehat{S}^{T^{(i)}_j}_{j,t} = \widehat{S}_{j,\min\{T^{(i)}_j,t\}} = 0$ means that j has not survived until $\min\{T^{(i)}_j, t\}$ and hence it also not present at the later or equal time $\min\{\max\{T^{(1)}_j T^{(2)}_j\}, t\}$. Hence $\widehat{S}^{\max\{T^{(1)}_j,T^{(2)}_j\}}_{j,t} = 0$ and we are done $\square$

PROOF OF THEOREM 6. $\widehat{S}^{T_{k_3}}_{\{k_1,k_2\},t} > 0$ only if $(k_1,k_2,k_3) \in \triangle_t$, and by Theorem 4 has unit expectation. $\square$

PROOF OF THEOREM 7. Distributing the covariance over the sum $\widetilde{N}_t(\triangle)$ in Theorem 6, $\mathrm{Var}(\widetilde{N}_t(\triangle))$ has unbiased estimator

$$\sum_{(k_1,k_2,k_3)\in\triangle_t} \widehat{S}^{T_{k_3}}_{\{k_1,k_2\},t}(\widehat{S}^{T_{k_3}}_{\{k_1,k_2\},t} - 1) + 2\sum_{\substack{(k_1,k_2,k_3)\prec \\ (k'_1,k'_2,k'_3)}} C^{T_{k_3},T_{k'_3}}_{\{k_1,k_2\},\{k'_1,k'_2\},t} \quad (23)$$

Where $\prec$ denotes $k_3 < k'_3$ in arrival order. Each variance term is of the form $1/q(1/q-1)$ for $q = p_{k_1,T_3}p_{k_2,T_3}$. Each covariance term is zero unless $\{k_1,k_2\} \cap \{k'_1,k'_2\} \neq \emptyset$. (The sets cannot be equal for then $k_3 = k'_3$ if both form triangles). The stated form the follows by rewriting the sum of covariance terms in (23) as a sum over edges $k \in \widetilde{K}_t$ of the covariances of all pairs snapshots that contain k as a sampled edge. $\square$

10. REFERENCES

[1] C. Aggarwal, Y. Zhao, and P. Yu. Outlier detection in graph streams. In *ICDE*, pages 399–409, 2011.

[2] N. K. Ahmed, N. Duffield, J. Neville, and R. Kompella. Graph sample and hold: A framework for big-graph analytics. In *SIGKDD*, 2014.

[3] N. K. Ahmed, J. Neville, and R. Kompella. Network sampling: From static to streaming graphs. *In TKDD*, 8(2):1–56, 2014.

[4] N. K. Ahmed, J. Neville, R. A. Rossi, and N. Duffield. Efficient graphlet counting for large networks. In *ICDM*.

[5] N. K. Ahmed, T. Willke, and R. A. Rossi. Estimation of local subgraph counts. In *IEEE Big Data*, pages 1–10, 2016.

[6] L. Becchetti, P. Boldi, C. Castillo, and A. Gionis. Efficient semi-streaming algorithms for local triangle counting in massive graphs. In *Proc. of KDD*, pages 16–24, 2008.

[7] A. Buchsbaum, R. Giancarlo, and J. Westbrook. On finding common neighborhoods in massive graphs. *Theoretical Computer Science*, 299(1):707–718, 2003.

[8] L. Buriol, G. Frahling, S. Leonardi, A. Marchetti-Spaccamela, and C. Sohler. Counting triangles in data streams. In *PODS*.

[9] N. Chiba and T. Nishizeki. Arboricity and subgraph listing algorithms. *SIAM J. on Computing*, 14(1):210–223, 1985.

[10] E. Cohen, N. Duffield, H. Kaplan, C. Lund, and M. Thorup. Efficient stream sampling for variance-optimal estimation of subset sums. *SIAM J. Comput.*, 40(5):1402–1431, Sept. 2011.

[11] E. Cohen and H. Kaplan. Summarizing data using bottom-k sketches. In *PODC*, 2007.

[12] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms*. 2nd edition, 2001.

[13] G. Cormode and S. Muthukrishnan. Space efficient mining of multigraph streams. In *PODS*, pages 271–282, 2005.

[14] L. De Stefani, A. Epasto, M. Riondato, and E. Upfal. Tri\est: Counting local and global triangles in fully-dynamic streams with fixed memory size. *In KDD*, 2016.

[15] N. Duffield, C. Lund, and M. Thorup. Learn more, sample less, control of volume and variance in network measurement. *IEEE T. Inf. Theory*, 51(5):1756–1775, 2005.

[16] N. Duffield, C. Lund, and M. Thorup. Priority sampling for estimation of arbitrary subset sums. *JACM*, 54(6):32, 2007.

[17] P. S. Efraimidis and P. G. Spirakis. Weighted random sampling with a reservoir. *IPL*, 97(5):181–185, 2006.

[18] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang. On graph problems in a semi-streaming model. *Theoretical Computer Science*, 348(2):207–216, 2005.

[19] M. R. Henzinger, P. Raghavan, and S. Rajagopalan. Computing on data streams. *External memory algorithms*, 50:107–118, 1998.

[20] M. Jha, C. Seshadhri, and A. Pinar. A space efficient streaming algorithm for triangle counting using the birthday paradox. In *In ACM SIGKDD*, pages 589–597, 2013.

[21] D. E. Knuth. *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*. Addison-Wesley, 1997.

[22] J. Leskovec, L. A. Adamic, and B. A. Huberman. The dynamics of viral marketing. *In TWEB*, 1(1):5, 2007.

[23] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, and N. Glance. Cost-effective outbreak detection in networks. In *KDD*, pages 420–429, 2007.

[24] Y. Lim and U. Kang. Mascot: Memory-efficient and accurate sampling for counting local triangles in graph streams. In *Proc. of SIGKDD*, pages 685–694. ACM, 2015.

[25] A. McGregor. Graph mining on streams. *Encyclopedia of Database Systems*, pages 1271–1275, 2009.

[26] S. Muthukrishnan. *Data streams: Algorithms and applications*. Now Publishers Inc, 2005.

[27] A. Pavan, K. Tangwongsan, S. Tirthapura, and K.-L. Wu. Counting and sampling triangles from a graph stream. *Proc. of VLDB*, 6(14):1870–1881, 2013.

[28] B. Rosén. Asymptotic theory for successive sampling with varying probabilities without replacement, I. *The Annals of Mathematical Statistics*, 43(2):373–397, 1972.

[29] R. A. Rossi and N. K. Ahmed. The network data repository withinteractive graph analytics and visualization. In *AAAI*, 2015.

[30] A. D. Sarma, S. Gollapudi, and R. Panigrahy. Estimating PageRank on Graph Streams. In *PODS*, pages 69–78, 2008.

[31] M. J. Schervish. *Theory of Statistics*. Springer, 1995.

[32] Y. Tillé. *Sampling Algorithms*. Springer-Verlag.

[33] J. S. Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software*, 11:37–57, 1985.

[34] D. Williams. *Probability with Martingales*. Cambridge University Press, 1991.

[35] P. Zhao, C. Aggarwal, and G. He. Link prediction in graph streams. In *ICDE*, pages 553–564, 2016.