PES: Priority Edge Sampling in Streaming Triangle Estimation

Roohollah Etemadi, Jianguo Lu,
School of Computer Science, University of Windsor, Canada
{etemadir, jlu}@uwindsor.ca

Abstract—The number of triangles ($\triangle$) is an important metric to analyze massive graphs. It is also used to compute clustering coefficient in networks. This paper proposes a new algorithm called PES (Priority Edge Sampling) to estimate triangles in the streaming model where we need to minimize the memory window. PES combines edge sampling and reservoir sampling. Compared with the state-of-the-art streaming algorithms, PES outperforms consistently. The results are verified extensively in 48 large real-world networks in different domains and structures. The performance ratio can be as large as 11. More importantly, the ratio grows with data size almost exponentially. This is especially important in the era of big data–while we can tolerate existing algorithms for smaller datasets, our method is indispensable in very large data sampling. In addition to empirical comparisons, we also proved that the estimator is unbiased, and derived the variance.

Index Terms—Graph sampling; Triangles; Streaming algorithms; Variance.

I. INTRODUCTION

The number of triangles (hereafter denoted as $\triangle$) is an important metric to reveal the complex structure of real-world networks. It has been used in many applications including community structure detection and graph clustering [1], link prediction [2], spam detection [3], DNA sequence analysis [4], microarray data analysis [5], word-learning [6], and many others. Exact algorithms to compute $\triangle$ in a large network are costly. It was proven that the best algorithm has a complexity of $\Theta(M^{3/2})$, where M is the number of edges in the input graph [7]. Therefore, various sampling-based algorithms are proposed, e.g., in [8]–[19].

Sampling-based algorithms are especially important in the era of big and hidden data. There are numerous massive networks that have billions of nodes. For example, Facebook as an online social network has over two billion users. Many networks are dynamic, both users and connections between users can change over time. Furthermore, networks are often hidden behind access interfaces, and data in its entirety are not available. Therefore, it is essential to design sampling-based methods.

There are two types of methods that estimate triangles and the closely related metric clustering coefficient. One is the direct-sampling that has random access to the nodes/edges of the input graph [8] [9] [20]. The other is the streaming model that scans the nodes/edges of the input graph in an arbitrary order over a stream. In the streaming model, a constant number of passes over the stream are used to estimate $\triangle$. The key constraint in streaming models is a limited memory window

[10] [11] [16]–[18] [21]. When there is no limit to the number of passes, it is called a semi-streaming model [22]. This paper addresses the estimation of $\triangle$ in the streaming model.

This paper proposes a new streaming algorithm, called PES (Priority Edge Sampling). It is based on edge sampling, and gives higher priority to edges that can form triangles. We prove that our estimator is unbiased, and derive the variance of the estimator so that the confidence interval can be obtained when an estimation is given. Empirically, we compare it with the state-of-the-art GPS-In algorithm [10], and demonstrate that PES outperforms GPS-In consistently on most of the 48 real networks that we have experimented with. More importantly, the performance gain increases with the size of networks. The performance ratio can be as high as 11, meaning that GPS-In needs 11 times more samples to achieve the same accuracy.

Performances of sampling algorithms are often data dependent, especially on the structure of the graphs. To verify our result in addition to empirical comparisons, we conduct analytical comparisons. GPS-In cannot give an analytical variance of the estimation because its sampling probability changes in every step. Hence, the comparison between PES and GPS-In cannot be analytical. To understand the advantage of PES, we compare it with NES (Naive Edge Sampling) that was proposed in [8] [13] [15]. The analytical comparison between PES and NES can shed some lights on understanding the difference between PES and GPS-In.

To summarize, our main contributions are that we have: 1) Given an efficient algorithm PES. 2) Proved the unbiasedness of the estimator and derived variances for PES and NES; 3) Compared PES and NES analytically.

II. BACKGROUND AND RELATED WORK

Given a simple graph $G(\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ stands for the set of nodes, and $\mathcal{E}$ the set of edges. Let $N = |\mathcal{V}|$, $M = |\mathcal{E}|$; $\triangle$ and Λ denote the number of triangles and wedges in G, respectively. Suppose that $1, 2, .., $ and M be the labels of the edges in $\mathcal{E}$ according to their arrival times in a stream. E.g., edge 1 arrives at time 1 and so on. A *wedge* $\mathcal{W}$ is a path (u, v, w) of length two, where $u, v, w \in \mathcal{V}$, $(u, v) \in \mathcal{E}$, and $(v, w) \in \mathcal{E}$. The wedge $\mathcal{W}$ is closed if $(u, w) \in \mathcal{E}$. Otherwise it is open. A closed wedge $\mathcal{W}$ is also called a triangle. Note that each triangle has three closed wedges. Table I summarizes the list of the notations used in the rest of this paper.

Each sampling method takes some sample nodes or edges, or a combination of them, into a subgraph. Then, the number

TABLE I: Summary of the notations

Notation	Meaning
$G(\mathcal{V}, \mathcal{E})$	Input graph (undirected and no self-edges)
g	A subgraph of G
N, M	Number of nodes and edges in G
p, q	Sampling probability
Λ	# wedges in G
Δ	# triangles in G
σ	A wedge pool
n	Size of pool σ
m	Sample size
Δ_σ	# triangles based on pool σ.
Λ_c	# candidate wedges identified based on g
Δ_g	# triangles based on g.
Φ	# pairs of shared triangles in G
$\widehat{\Delta}_{NES}$	Naive edge sampling estimator
$\widehat{\Delta}_{PES}$	Priority edge sampling estimator

of triangles in the subgraph is used to estimate the triangle count. Depending on the way to take samples, the estimator and its variance change. Intuitively, we want to observe a maximum number of triangles while keeping the sample size small. The bottom line is that we need to observe at least one triangle in order to give an estimate.

A. Node-based methods

The most naive method of triangle estimation is to sample three random nodes as a potential triangle, then check the existence of edges among the nodes over a stream. It is called *triple sampling* [17]. This approach needs to sample $4N^3/(6\epsilon^2\Delta)$ number of triples to achieve an estimation in interval $\Delta \pm \epsilon\Delta$ with %95 confidence. Intuitively, the complexity of three nodes combination is $O(N^3)$. Obviously, it is not a practical method because the sample size is too large to observe even one triangle. The cost is even higher than direct counting of the triangles.

A more practical method is to sample a random edge and a random node [16], then check whether they form a triangle. This method improves the previous triple sampling by assuming one edge always exists in the triple. Hence, it only needs to check the existence of other two edges. Still, it needs to take $4MN/(3\epsilon^2\Delta)$ triples to have an estimation in the same confidence interval as in triple sampling.

Large real networks are mostly sparse, hence the probability of having a triangle is still low among two random pairs of nodes. One improvement to the above method is, instead of choosing a random node in the entire graph, selecting a random node from its neighbourhood. It is called *neighborhood sampling* [14] [18].

B. Edge-based methods

The most straightforward edge sampling is to take edges uniformly at random, then count the triangles in the subgraph [9]. In the streaming model, the corresponding streaming version of the algorithm is to take each edge with an equal probability p over a stream and create a subgraph g. The number of triangles in g is used to estimate Δ. Obviously, the sampling probability of a triangle in such method is p^3. The size of g needs to be $1.5M/(\epsilon^2\Delta)^{1/3}$ to obtain an estimation

with an additive error $\pm\epsilon\Delta$ with %95 confidence. When p is small, which is the case for very large graphs, this algorithm is not efficient.

Instead of using equal probability among three edges, there are methods to assign high probabilities for the second and/or the third edge. For instance, *post-stream priority sampling* (GPS-Post) [10] takes this approach by sampling the third edge with a higher probability.

Another technique is to take edges from the neighborhood of already sampled edges with higher probability. A pair of connected edges (called a wedge) in the sample can be a potential triangle, and its closeness is checked in the rest of the stream [12] [13] [15]. Obviously, the probability of forming a wedge is p^2 because its two edges are required to be sampled. [11] improves the previous method as follows. When an edge closes a wedge in a sample it is unconditionally added into the sample; if it is connected to some sampled edges it is chosen with higher probability q; otherwise it is taken with probability p. The number of triangles in the sample is used to estimate Δ. Obviously, this method samples triangles with different probabilities, i.e., pq, q^2, p, q, and 1. One shortcoming of this approach is that how one can determine q - sampling probability of a neighbor edge. To overcome such an issue, in our method q is dynamically adjusted using *reservoir sampling* [23].

More recently, another elegant approach has been proposed by [10] called *in-stream priority sampling* (GPS-In). It preserves edges in a sample with different priorities. The number of sampled wedges closed by an edge is used as a measure to determine the priority of the edge being preserved in the sample. For each new edge e, it first counts the number of wedges closed by e in the sample and computes its priority. Then, the edge is added into the sample. If the number of edges in the sample exceeds the size limit, an edge with lower priority is removed from the sample. In each step, the estimator for Δ is updated if edge e completes some wedges in the sample. It has been shown that GPS-In outperforms the existing methods [10]. Therefore, we consider GPS-In as the state-of-the-art method in this context.

When random access to the input graph is available the ideal method is *wedge sampling*. It selects some wedges uniformly at random and checks their closeness to estimate Δ. Unfortunately, taking a wedge uniformly at random in a large graph is costly. Three passes over an edge stream are required to implement wedge sampling in the streaming model [16]–[18].

Another direction is indirect sampling. Such methods have been applied when the entire graph is not accessible. They use traversal-based sampling techniques to take a sample from the input graph [24] [25]. Moreover, several works have been conducted to compute clustering coefficient closely related to Δ [25]–[28].

III. NAIVE EDGE SAMPLING (NES)

As a starting point for understanding our PES algorithm to be described in the next section, we first present a naive algorithm based on edge sampling, called NES (Naive Edge

Sampling). It is similar to TRIEST [13] and MASCOT [15]. The details of NES are shown in Algorithm 1. For each edge in a stream, NES adds the edge into subgraph g with probability p (Line 4). Then, the same edge is used to check how many wedges in current g are closed by it. Δ_g records the sum of such closed wedges(triangles) (Lines 5-7).

The algorithm differs from the one in [8] in that we do not count the triangles in g. Instead, it checks the closeness of wedges in g during the streaming process. Clearly, the probability of forming a wedge in g is p^2. Note that three edges of a closed wedge can appear in six different orders in a stream. In two of them, the third edge appears after the first two and the associated closed wedge can be observed. Thus, the probability of identifying a closed wedge is $p^2/3$. Because each triangle has three closed wedges, the sampling probability of each triangle is $p^2/3 \times 3 = p^2$. Note that each identified closed wedge by NES is considered as a one triangle because only one of three closed wedges of each triangle can be identified in a stream.

Suppose δ_i be an indicator for the ith triangle in the original graph G. Indicator δ_i is one when the ith triangle is identified over the stream; otherwise it is zero. Recall that Δ_g is the number of triangles identified by NES based on g over a stream. The expectation of Δ_g is $\mathbb{E}(\Delta_g) = \mathbb{E}(\sum_{i=1}^{\Delta} \delta_i) = \sum_{i=1}^{\Delta} \mathbb{E}(\delta_i) = \sum_{i=1}^{\Delta} p^2 = p^2\Delta$. Thus, the unbiased estimator for Δ using NES is $\widehat{\Delta}_{NES} = \frac{\Delta_g}{p^2}$.

Algorithm 1: Naive Edge Sampling (NES)

Input: p
Output: $\widehat{\Delta}$, RSE($\widehat{\Delta}$)
1 **begin**
2    $\Delta_g = 0$, $g = \{\phi\}$.
3   **while** *new edge e* **do**
4   Add e into g with probability p.
5   **foreach** *wedge $w \in g$ closed by e* **do**
6        $\Delta_g += 1$.
7   **end**
8   **end**
9    $\widehat{\Delta}_{NES} = \Delta_g/p^2$.
10   RSE($\widehat{\Delta}_{NES}$) $\approx \Delta_g^{-1/2}$.
11 **end**

Next we need to understand the variance of $\widehat{\Delta}_{NES}$. Although MASCOT gave a similar algorithm, they only give upper-bound of its variance. We derived the variance of $\widehat{\Delta}_{NES}$ and present it in the form of Relative Standard Error (RSE=$\sqrt{var}/\Delta$) in Theorem 1. We use RSE instead of variance that is commonly used. This is because variance depends on the ground truth, which changes from data to data. This is especially inconvenient when evaluating multiple data sets– a larger variance in one data may be better than a smaller variance in another data. The variance of NES is adapted but different from the direct sampling algorithm in [8] to accommodate the streaming model. The main difference is that in NES, to identify a closed wedge over a stream, first its two edges need to be added into g; then its third edge needs to be visited in the rest of the stream.

Theorem 1. *The RSE of $\widehat{\Delta}_{NES}$ is approximated by*

$$RSE(\widehat{\Delta}_{NES}) \approx \Delta_g^{-1/2}. \qquad (1)$$

Proof. See Appendix A.

Theorem 1 shows that the variance depends on the number of triangles in the sampled graph g. To reduce RSE with the same subgraph size g, we need to sample more triangles while keeping the same sampling probability for the first edge. This prompts us to increase the sampling probability for the second edge of a triangle.

IV. PRIORITY EDGE SAMPLING (PES)

A. The algorithm

PES improves NES by increasing the probability of capturing triangles in the sample graph. To do so, we maintain a pool of wedges as well as a subgraph g. Edges that can form a wedge in g will have a higher priority being sampled. Hence, we call it Priority Edge Sampling. It is impossible and not necessary to keep all the wedges. Instead, we maintain a small fixed-size pool of wedges σ. For each triangle, the first edge will be sampled with probability p, which is the same as NES. The difference is in the second edge. When the second edge is scanned, the associated wedges are added into σ with probability q. Later we will show that q is normally much larger than p, especially when the graph is large. The closeness of wedges in the pool is checked in the rest of the stream. Therefore, PES identifies a triangle with probability pq, which is greater than p^2 in NES.

The details of PES are summarized in Algorithm 2. Input p is the sampling probability of edges, n is the pool size. In our experiments, we simply set $n = |g|$ for the convenience of performance comparison. Λ_c counts the wedges formed in g. Some of these wedges may be added to σ with a changing probability q. Hence we call them *candidate* wedges and denoted by Λ_c. Δ_σ counts the triangles formed from σ and g. When a new edge e is visited, it is added into subgraph g with probability p (Line 4). Then, the closeness of wedges in pool σ is checked (Lines 5-8). Once a closed wedge is identified, the number of triangles Δ_σ captured so far is increased by 1 (Line 7). Next, each candidate wedge formed using the new edge e and edge f in g like $w(e, f)$ is considered to be added into pool σ with probability q (Lines 9-21). Note that probability q is dynamically computed over the stream using n and Λ_c (Line 14). We explain the steps in the following illustrative example.

B. Example

We illustrate PES with a toy graph in Fig. 1 with detailed steps. Each row in the table represents one step. Column e shows the edge stream. Column g displays the sampled edges in subgraph g. In this example, each edge in the stream is added into g with probability $p = 0.2$. When edge $(1, 4)$ arrives, PES adds it to g with probability p. Suppose that it is not added, and g remains empty. Next edge in the stream is $(6, 8)$. Suppose that it is added to g this time. It can not form any wedges in the fourth column.

Algorithm 2: Priority Edge Sampling (PES)

Input: p, n.
Output: $\widehat{\Delta}$, RSE($\widehat{\Delta}$)

1 **begin**
2    $\Lambda_c = 0$, $\Delta_\sigma = 0$, $\sigma = \{\phi\}$, $g = \{\phi\}$.
3   **while** *new edge e* **do**
4   Add edge e into g with probability p;
5   **foreach** *wedge w in σ closed by e* **do**
6   label w as closed.
7        $\Delta_\sigma += 1$.
8   **end**
9   **foreach** *wedge $w(e,f)$ where edge $f \in g$* **do**
10        $\Lambda_c += 1$.
11   **if** $|\sigma| < n$ **then**
12          $\sigma = \sigma \cup \{w\}$.
13   **else**
14          $q = n/\Lambda_c$.
15   **if** *Random[0,1]$< q$* **then**
16   Select random wedge w' from σ.
17   **if** w' *is closed* **then** $\Delta_\sigma -= 1$.
18            $\sigma = \sigma - \{w'\}$.
19            $\sigma = \sigma \cup \{w\}$.
20   **end**
21   **end**
22   **end**
23    $\widehat{\Delta}_{PES} = \Delta_\sigma / pq$.
24   RSE($\widehat{\Delta}_{PES}$)$\approx \Delta_\sigma^{-1/2}$.
25 **end**

(A) An example graph.

	e	g	$w(e,f), f \in g$	σ	Λ_c	q	Δ_σ
1	(1,4)	ϕ	-	ϕ	0	-	0
2	(6,8)	**(6,8)**	-	ϕ	0	-	0
3	(6,7)	(6,8)	(7,6,8)	$(7,6,8)^-$	1	1	0
4	(1,6)	(6,8)	(1,6,8)	$(7,6,8)^-,(1,6,8)^-$	2	1	0
5	(6,11)	(6,8)	(8,6,11)	$(7,6,8)^-,(1,6,8)^-$	3	0.66	0
6	(2,3)	(6,8)	-	$(7,6,8)^-,(1,6,8)^-$	3	0.66	0
7	(9,10)	(6,8)	-	$(7,6,8)^-,(1,6,8)^-$	3	0.66	0
8	(1,2)	(6,8),**(1,2)**	-	$(7,6,8)^-,(1,6,8)^-$	3	0.66	0
9	(6,10)	(6,8),(1,2)	(8,6,10)	$(8,6,10)^-,(1,6,8)^-$	4	0.5	0
10	(1,5)	(6,8),(1,2)	(2,1,5)	$(8,6,10)^-,(1,6,8)^-$	5	0.4	0
11	(6,9)	(6,8),(1,2)	(9,6,8)	$(8,6,10)^-,(9,6,8)^-$	6	0.33	0
12	(1,3)	(6,8),(1,2)	(2,1,3)	$(2,1,3)^-,(9,6,8)^-$	7	0.28	0
13	(8,9)	(6,8),(1,2)	(6,8,9)	$(2,1,3)^-,(9,6,8)^+$	8	0.25	1

(B) Steps on the graph in Panel (A) with $p = 0.2, n = 2$.
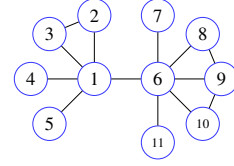
Fig. 1: Steps of applying our PES on a toy graph.

The third edge $(6,7)$ is not added into g, but we still check its neighbours in g for closed wedges and *candidate* wedges. The *candidate* wedges constructed in each step are demonstrated in the fourth column. When edge $(6,7)$ is encountered in step 3, a wedge $(7,6,8)$ is formed since edge $(6,8)$ is already in the subgraph g. In the pool for each wedge, we keep a label to show its closeness. The open wedge $(7,6,8)$ is denoted as $(7,6,8)^-$. Column Λ_c records the number of such candidate wedges. It can be larger than the pool size. When edge $(6,11)$ arrives, it forms a candidate wedge $(8,6,11)$, hence Λ_c is increased by one, but it is not added into the pool σ.

Not every candidate wedge is added into the pool. The pool has a fixed size, functioning as a reservoir. In this example, its capacity $n = 2$. The candidate wedge is added into the pool unconditionally only when it is not full yet. Hence, wedge $(7,6,8)$ and the wedge in the subsequent step $(1,6,8)$ are added into the pool.

When the pool is full, the candidate wedge will replace a random wedge in the pool with probability q. In step 9, edge $(6,10)$ forms a candidate wedge $(8,6,10)$ with edge $(6,8)$. Now the forth wedge $(8,6,10)$ can not be added into σ directly because the pool has reached its limit 2. Instead, we replace one of the wedges in the pool with a probability $q = n/\Lambda_c = 2/4$. Suppose that by chance, this wedge replaces $(7,6,8)$ in the pool. The candidate wedge in Step 10 does not replace any wedge in the pool by chance. For the candidate wedge $(9,6,8)$ in step 11, suppose that it replaces an existing wedge $(1,6,8)$ in the pool. Step 12 has another wedge being replaced.

The last edge in the stream is $(8,9)$. It closes the wedge $(9,6,8)^-$ that is obtained in previous steps. Hence, the label

of this wedge is changed to $+$; and Δ_σ is increased by 1. At this point, $\Lambda_g = 8$. This means eight candidate wedges are identified in total over the stream; the probability of preserving a wedge in σ is $q = 2/8$. Thus, the unbiased estimator for Δ is

$$\widehat{\Delta}_{PES} = \frac{\Delta_\sigma}{pq} = \frac{1}{0.2 \times 0.25} = 20. \quad (2)$$

C. The unbiased estimator

We prove that $\widehat{\Delta}_{PES}$ is unbiased as follows. Let δ_i be the indicator function for the ith triangle in the input graph. It is one when the ith triangle is sampled; otherwise it is zero. For each triangle, the probability of sampling the first edge is p, the probability of sampling the second edge is q. Note that the closeness of a wedge is checked every time when a wedge emerges in the pool. Hence the probability of sampling a triangle is pq. The expectation of δ_i is pq and the expectation of Δ_σ is

$$\mathbb{E}(\Delta_\sigma) = \mathbb{E}(\sum_{i=1}^{\Delta} \delta_i) = \sum_{i=1}^{\Delta} \mathbb{E}(\delta_i) = \sum_{i=1}^{\Delta} pq = pq\Delta. \quad (3)$$

Thus, the unbiased estimator is as follows.

Theorem 2. *The unbiased estimator for PES algorithm is*

$$\widehat{\Delta}_{PES} = \frac{\Delta_\sigma}{pq}. \quad (4)$$

An interesting part of the algorithm is that q decreases over time, and the sampling probability of the second edge in Eq. 4 is the q in the final step, not the bigger q values in earlier steps. Intuitively, edges sampled in earlier steps have a higher

probability of being replaced during the process. The earlier the edge being scanned, the bigger the q is at that moment. But it also has a higher probability being replaced in a later stage. Hence the overall probability is the same as the final q. Detailed proof is similar to reservoir sampling [23] using inductive inference, and is given as follows.

For the last candidate wedge at arrival time Λ_c, it is easy to understand that the second edge has a sampling probability $q = n/\Lambda_c$. Other wedges arrived before also has a sampling probability q, following reservoir sampling [23] as explained in the following inductive inference:

When $\Lambda_c = n + 1$, the sampling probability for wedges arrived before time n is:

$$1 \times \left(\frac{1}{n+1} + \frac{n}{n+1} \frac{n-1}{n} \right) = \frac{n}{n+1}. \tag{5}$$

This is because that there is a probability of $1/(n+1)$ that the new wedge won't replace any old wedge; and there is a probability of $n/(n+1)$ that an old wedge will be replaced. For each replacement, the probability of one particular wedge not being replaced is $n - 1/n$.

Suppose that the old wedges are kept with probability $n/(n + x)$ when $\Lambda_c = n + x$. When $\Lambda_c = n + x + 1$, the sampling probability for wedges arrived before time $n+x+1$ is

$$\frac{n}{n+x} \times \left(\frac{x+1}{n+x+1} + \frac{n+x}{n+x+1} \frac{n-1}{n+x} \right) = \frac{n}{n+1}. \tag{6}$$

D. The variance

The variance of the estimator is complicated because of the involvement of two different sampling techniques–uniform sampling and reservoirs sampling. In PES, a wedge as a possible triangle is formed uniformly at random with probability p over an edge stream; and it is preserved with probability q in pool σ. Applying the variance on the estimator we get

$$var(\widehat{\Delta}_{PES}) = var\left(\frac{\Delta_\sigma}{pq} \right) = var\left(\sum_{i=1}^{\Delta} \frac{\delta_i}{pq} \right)$$

$$= \frac{1}{(pq)^2} \sum_{i=1}^{\Delta} \sum_{j=1}^{\Delta} cov(\delta_i, \delta_j)$$

$$= \frac{1}{(pq)^2} \left(\sum_{i=1}^{\Delta} var(\delta_i) + \sum_{i \neq j}^{\Delta} cov(\delta_i, \delta_j) \right). \tag{7}$$

Recall that δ_i is the indicator for the ith triangle as defined before. By the definition of variance, $var(\delta_i)$ is $\mathbb{E}(\delta_i) - \mathbb{E}(\delta_i)\mathbb{E}(\delta_i)$. Therefore, the cost of the first term in Eq. 7 is $\Delta(pq - (pq)^2)$. For the covariance, let Φ be the number of pairs of triangles with a common edge. To identify such a case by PES, the common edge should be added into g with probability p. Furthermore, the other two edges need to be preserved in pool σ with probability $(n^2 - n)/(p^2\Lambda^2 - p\Lambda)$. Thus, the probability of sampling such a dependent pair is pq'^2 where q'^2 is $(n^2-n)/(p^2\Lambda^2-p\Lambda)$. Recall that n is the size of pool σ. Each dependent pair has five edges and the common edge should be visited before the other four and needs to be sampled with probability p. Clearly, the five edges can arrive

in 120 different orders in a stream; and in one-fifth of them, the common edge is the first one in the stream. Note that each dependent pair (δ_i, δ_j) appears twice in the covariance term. Thus, the cost of Φ dependent cases is $\frac{2\Phi}{5}(pq'^2 - (pq)^2)$. Because the reservoir sampling is used to preserve wedges in pool σ we need to consider the cost of $(\Delta^2 - 2\Phi - \Delta)$ independent pairs. Obviously, the probability of selecting a pair of independent triangles is $p^2q'^2$. By the definition of covariance, i.e. $\mathbb{E}(\delta_i\delta_j) - \mathbb{E}(\delta_i)\mathbb{E}(\delta_j)$, the cost of independent cases is $(\Delta^2 - 2\Phi - \Delta)(p^2q'^2 - (pq)^2)$. Substitute the costs in Eq. 7 and after some math simplification, the variance of the estimator is given by the following theorem.

Lemma 1. *Let Δ be the true number of triangles and $\widehat{\Delta}_{PES}$ be its estimation by PES. The variance of $\widehat{\Delta}_{PES}$ is*

$$var(\widehat{\Delta}_{PES}) = \frac{\Delta(1 - pq)}{pq} + \frac{2\Phi(q'^2 - pq^2)}{5pq^2} + \frac{\Phi'(q'^2 - q^2)}{q^2}. \tag{8}$$

here Φ is the number of pairs of shared triangles and $q = n/p\Lambda$ and $q'^2 = (n^2 - n)/(p^2\Lambda^2 - p\Lambda)$, and $\Phi' = (\Delta^2 - 2\Phi - \Delta)$.

The variance of the estimator depends on several metrics including Δ, Φ, p and q. In practice, we do not have the knowledge of these metrics. For example, Δ is exactly what we are estimating. Hence, in order to know the performance of the estimator, we need to estimate the variance. Thus, we simplify the variance to have better insight into it. To do so, we translate the variance into RSE and use big data assumption to present the following theorem.

Theorem 3. *The RSE of $\widehat{\Delta}_{PES}$ is approximated by*

$$RSE(\widehat{\Delta}_{PES}) \approx \Delta_\sigma^{-1/2}. \tag{9}$$

Proof. Translate $var(\widehat{\Delta}_{PES})$ into the RSE= $\sqrt{var}/\Delta$. When the input graph is large, approximations $n - 1 \approx n$ and $p\Lambda - 1 \approx p\Lambda$ are valid. Thus, after some math work we get

$$RSE(\widehat{\Delta}_{PES}) \approx \left[\frac{1}{\Delta pq} \left(1 - pq + \frac{2\Phi}{5\Delta}(q - pq) \right) \right]^{1/2}. \tag{10}$$

When graph is large sampling probabilities p and q is very small and terms $-pq$ and $+\frac{2\Phi}{5\Delta}(q-pq)$ in Eq. 10 are ignorable. Thus, Eq. 10 is simplified as $(\Delta pq)^{-1/2}$. Because $\Delta_\sigma = \Delta pq$, we obtain the theorem.

V. EXPERIMENTS

We conduct experiments to 1) compare our algorithms with the state-of-the-art algorithms GPS-In and GPS-Post [10]. Other algorithms are not compared because it is already demonstrated that they are inferior to GPS-In; and 2) Verify our analytical results presented in Theorem 1 and 3. This is needed because there are approximations in the derivation. The precise results are long formulas that depend on the structure of the graph, such as the number of triangles (Δ) and the count of dependent triangles (Φ). Theorems 1 and 3 give more concise results by omitting some terms in the long formula by assuming the graph is large and p is small. How good is such approximation needs to be evaluated empirically.

TABLE II: Properties of the networks in our experiments, sorted by graph size N.

Dataset	$N(\times 10^6)$	$\langle d \rangle$	$\mathcal{C}$	Type	Dataset	$N(\times 10^6)$	$\langle d \rangle$	$\mathcal{C}$	Type
1. Ego-facebook [29]	0.004	43.69	0.519	OSN[1]	25. Youtube [29]	1.1	5.27	0.006	OSN
2. CA-GrQc [29]	0.005	5.52	0.629	COL[2]	26. Dblp [30]	1.3	8.16	0.170	COA
3. Wiki-vote [29]	0.007	28.32	0.125	OSN	27. Wiki-Polish [30]	1.5	55.17	0.01	WEB
4. AstroPh [30]	0.01	21.10	0.31	CIT[3]	28. Trec-wt10g [30]	1.6	8.33	0.014	WEB
5. CA-CondMat [29]	0.02	8.08	0.264	COA[4]	29. Wiki-Portuguese [30]	1.6	48.19	0.022	WEB
6. HepPh [30]	0.02	224.14	0.279	COA	30. Wiki-Japanese [30]	1.6	69.82	0.021	WEB
7. Enron-email [30]	0.03	10.02	0.085	ECO[5]	31. Pokec [30]	1.6	27.31	0.046	OSN
8. Brightkite [29]	0.05	7.35	0.110	OSN	32. As-skitter [29]	1.6	13.08	0.005	INT[8]
9. Facebook [30]	0.06	25.64	0.147	OSN	33. Wiki-Italian [30]	1.8	72.90	0.024	WEB
10. Epinions [30]	0.07	10.69	0.065	OSN	34. Hudong [30]	1.9	14.54	0.003	WEB
11. Slashdot-Zoo [30]	0.07	11.82	0.023	OSN	35. Hollywood [31], [32]	1.9	24.51	0.152	OSN
12. Livemocha [30]	0.1	42.13	0.014	OSN	36. Flicker [30]	2.3	19.83	0.107	OSN
13. Douban [30]	0.1	4.22	0.01	OSN	37. Flixster [30]	2.5	6.27	0.013	OSN
14. Gowalla [29]	0.1	9.66	0.023	OSN	38. Wiki-Russian [30]	2.8	44.20	0.015	WEB
15. Libimseti [30]	0.2	155.97	0.007	OSN	39. Wiki-Franch [30]	3.0	55.21	0.015	WEB
16. Digg [30]	0.2	11.07	0.061	OSN	40. Orkut [30]	3.0	76.28	0.041	OSN
17. Dblp-Coau [29]	0.3	6.62	0.306	COA	41. Wiki-German [30]	3.2	40.77	0.0088	WEB
18. Web-NotreDame [29]	0.3	6.69	0.087	WEB[6]	42. USpatent [30]	3.7	8.75	0.067	CIT
19. Amazon [29]	0.3	5.53	0.205	COP[7]	43. LiveJournal [29]	3.9	17.35	0.125	OSN
20. Actor [30]	0.3	78.68	0.166	COL	44. DBpedia [30]	18	13.89	0.0016	WEB
21. Citeseer [30]	0.3	9.03	0.049	CIT	45. Web-Arabic [31], [32]	22	48.70	0.031	WEB
22. Dogster [30]	0.4	40.03	0.014	OSN	46. Gsh-2015 [31], [32]	29	9.18	0.007	WEB
23. Catster [30]	0.6	50.32	0.028	OSN	47. MicrosoftAc.G. [33]	46	22.61	0.015	CIT
24. Web-Google [30]	0.8	9.87	0.055	WEB	48. Friendster [30]	65	55.06	0.017	OSN

[1] Online Social Network   [2] Collaboration   [3] Citation   [4] Coauthorship   [5] E-communication   [6] Web Graph   [7] Co-purchasing   [8] Internet topology

The code along with all the data, including some intermediate data, are available at http://cs.uwindsor.ca/~etemadir/PES.

A. Data

Because the performance of sampling algorithms often varies from data-to-data, especially depends on the structure of the graphs, we verify our results extensively with many (48) real networks with different size from varieties of domains. The size ranges from 4 thousand to 65 million nodes. The domains include online social networks (OSN), web graphs, citation and co-authorship networks, etc. In some figures, we only plot half of the datasets (24) to save space. Other datasets have similar behaviours.

It is computationally costly to obtain the ground truth of large graphs. Luckily, we have access to two servers each with 24 cores and 256 GB RAM to carry out such intensive computing. Table II summarizes the networks and their statistics. The graphs are sorted by their node size N. In the table $\langle d \rangle$ is average degree, and $\mathcal{C}$ is clustering coefficient ($\mathcal{C} = 3\Delta/\Lambda$). We executed the estimators on the graphs and reported the results along with our observations in the following sections. The results were obtained over 1000 independent runs for the graphs except for the four largest graphs that are repeated 500 times.

B. Comparison with GPS-In and GPS-Post

Fig. 2 summarizes the comparison of PES with the state-of-the-art methods GPS-Post (Panel A) and GPS-In (Panel B) [10]. We also compared NES and our PES algorithm in Panel C. We set the sampling probability of the estimators to obtain the same RSE. Here we report the ratios between sample sizes when RSE=0.2. Similar phenomenon is observed for other RSEs. In each panel, the Y-axis is the ratios, and the X-axis is the graph size that is represented by the node size N multiplied by clustering coefficient. In all the methods, m is the 'sample size'. Algorithms differ in the definition of 'sample size' because some algorithms maintain a reservoir of wedges in addition to subgraph g or use extra memory per sampled edge to store information about sampled edges in subgraph g. NES has g only. Hence the sample size m is the number of edges (denoted by $|g|$), which is equal to pM. PES maintains a wedge pool σ. Hence the sample size is $|g| + |\sigma|$. GPS -In and -Post also store subgraph g and two additional values per each edge in g. However, we consider their sample sizes as $|g|$.

In the panels, each marker represents one of the 48 graphs described in Table II. From the figure we make several observations:

- Our PES outperforms GPS-In and GPS-Post consistently in terms of sample size. All the ratios are above one in Panel (A), meaning that PES needs fewer samples than GPS-Post for all the datasets. For instance, take Orkut (labeled 43) in Panel A has ratio 73, meaning that GPS-Post needs 73 times more sampled edges compared to our PES. Compare to GPS-In, our PES also needs less sample size in most of the graphs. For example, LiveJournal (labeled 43) in Panel A has the ratio 5.4, meaning that GPS-In requires 5.4 times more sampled edges compared to PES to obtain an estimation with the same RSE. The improvement margin is higher for GPS-Post, which is expected since GPS-In improves GPS-Post. Take the same LiveJournal data for example, as shown in Panel A, the ratio is 66, much higher than 5.4.
- The ratio grows almost exponentially with data size. In other words, compared with PES, the sample size of other algorithms grows exponentially with graph size. This result has high implication for very large graphs: although other algorithms can deal with current data, their performance will deteriorate exponentially with graph
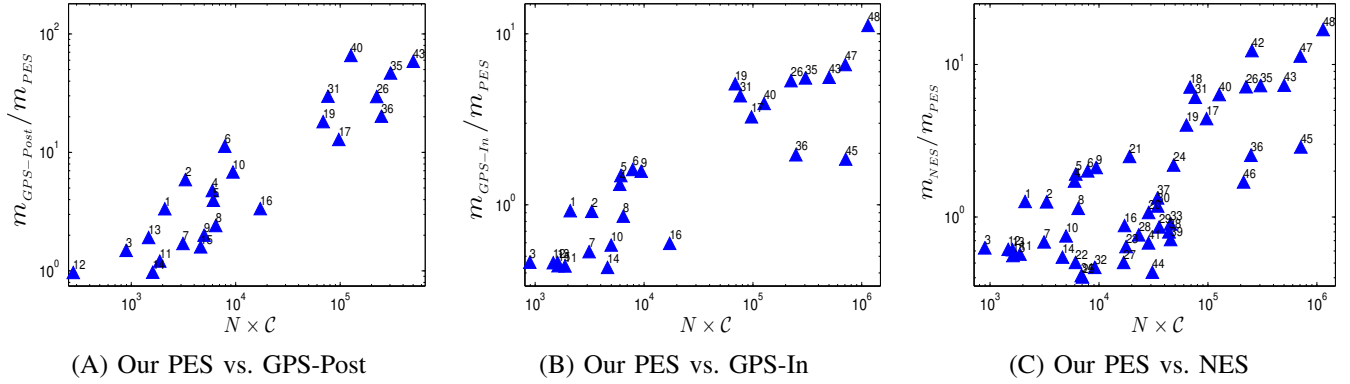
Fig. 2: Sample size ratios of our PES vs. GPS-Post (Panel A), GPS-In (Panel B), and NES (Panel C) when RSE=0.2.
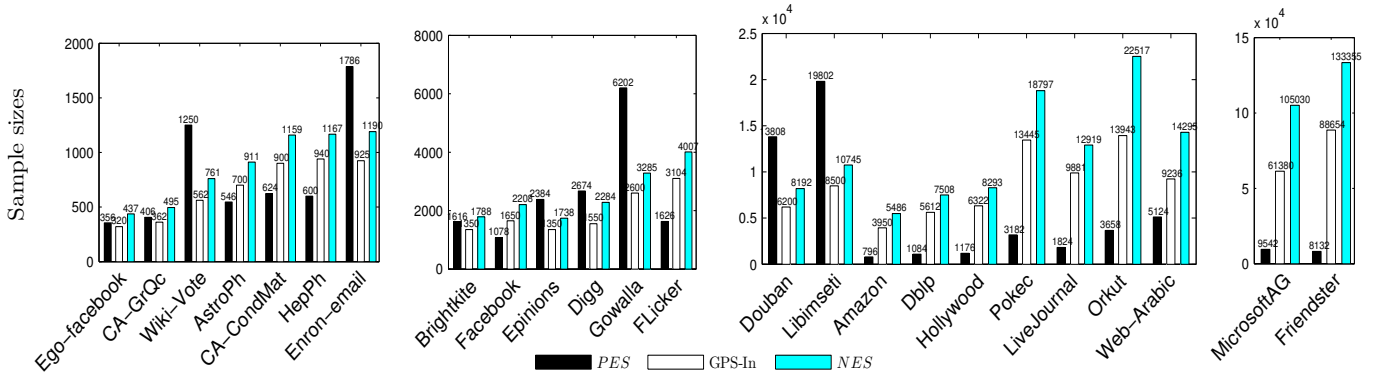

Fig. 3: Our PES uses **less sample sizes** compared to other methods to obtain an estimation with the same RSE=0.2 on most of the graphs. Note that the sample size include both the size of the subgraph and the reservoir for our PES but for GPS methods extra memory per sampled edge was ignored.

size. The Pearson correlation coefficient between the ratios and the size of data is 82 for GPS-In, 80 for GPS-Post, and 79 for NES.

Fig. 3 compares the actual sample sizes of the three methods side by side. The sample sizes are the ones to achieve the same RSE=0.2. Take the Friendster data for example, the samples for PES is 8,132, meaning that the subgraph size is 4,066, and the reservoir size is 4,066 to achieve RSE=0.2. On the other hand, the sample size of GPS-In is 88,654, meaning that $|g| = 88,654$. Similarly, the sample size of NES is 133,355, meaning that $|g| = 133,355$. Note that even though GPS-In outperforms NES in terms of $|g|$, the overall cost including both g and extra data for sampled edges in g is actually higher. As shown in the figure, our PES outperforms the other methods in most of the graphs. It is obvious that,the performance ratios grow by increasing the size of graphs. For example, all the methods need almost the same sample sizes for Ego-facebook graph (the smallest graph in our dataset) to achieve the estimation with the same RSE=0.2. Take the Friendster data as the largest graph in the dataset, PES needs 10.9 times less sample size compare to GPS-In.

Next, we investigate how the performance ratios between the methods change by increasing the accuracy of estimators (decreasing the RSE). To do so, we set the parameters of PES to obtain the RSEs between 0.1 and 0.4. Then, GPS-In was run using the same sample sized used in PES. Note that we

considered both the size of subgraph g and the pool σ as a sample size of PES. We only report the observed RSEs of our PES vs. GPS-In in Fig. 4 because GPS-In is more efficient compared to other existing methods. It can be seen that by increasing the sample size, the gap between the RSEs of the methods diminishes. Still, PES outperforms GPS-In in terms of obtaining accurate estimation using the same sample sizes for large graphs, as we can see in the last row of the plot.

C. Validation of Theorems 1 & 3

We conduct experiments to verify our approximations used in the derivations of Theorems 1 & 3. Thus, sampling probability p of the PES and NES were initialized in a way that the estimators achieve the RSEs between 0.1 and 0.4 to get estimations in range $[\Delta \pm 0.8\Delta, \ \Delta \pm 0.2\Delta]$ with 95% confidence. The observed and estimated RSEs are reported in the plots of Fig. 5 and 6. We report the results for 24 representative graphs. Similar patterns are observed for the remaining data sets. As shown in the plots, in both theorems our approximations work very well. It can be seen that our estimated RSEs (blue lines in the plots) fit perfectly the observed ones (red lines with circle markers) not only for large graphs but also for small-sized ones. Thus, in practice the theorems can be used to control the accuracy of the estimators. Moreover, they can be used to quantify the performance ratio between the methods as in the following section.
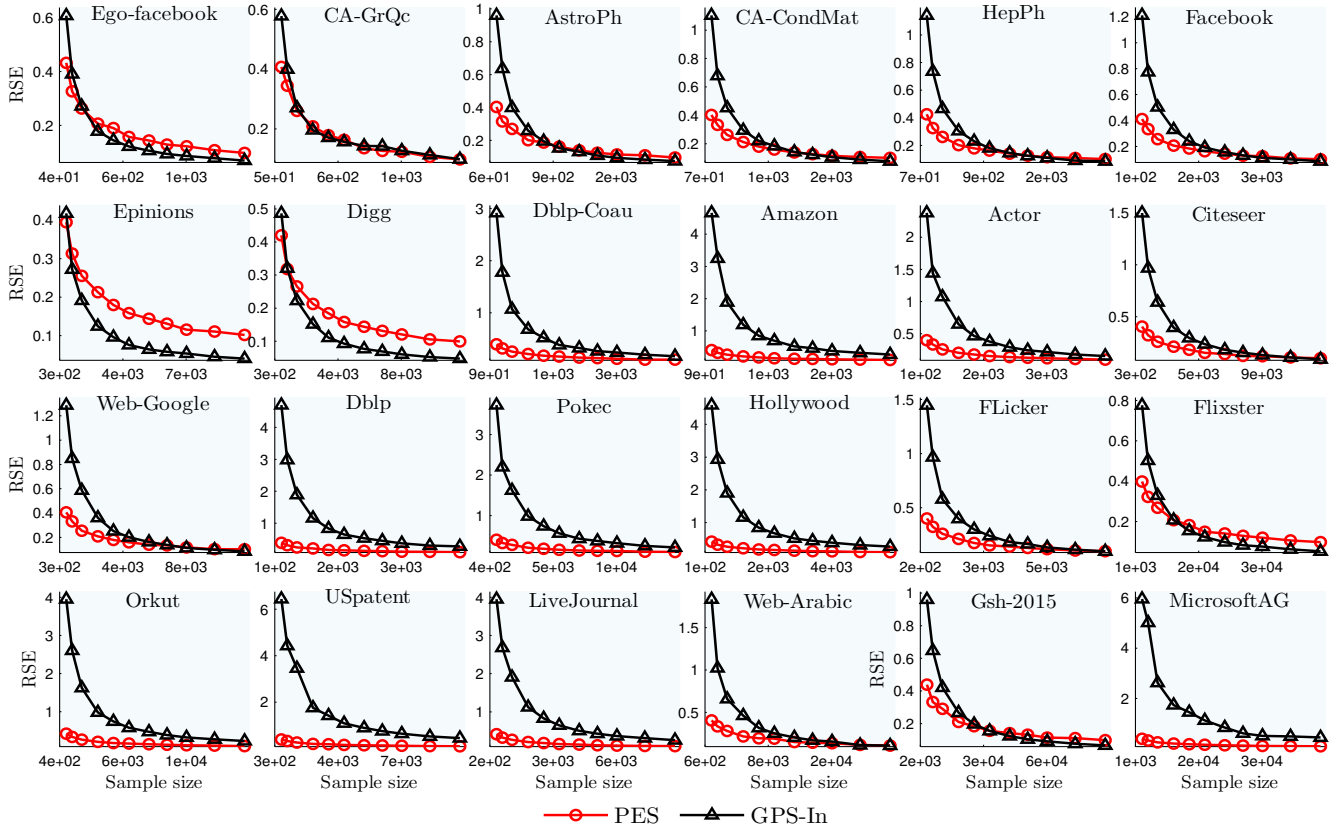
Fig. 4: Our PES outperforms GPS-In in terms of RSEs when both the methods are using the same sample sizes. Note that for our PES, the sample size includes both the size of subgraph g and reservoir size, i.e., $|\sigma|$.

D. An implication of Theorems 1 & 3

We use Theorems 1 & 3 to quantify the performance ratio between NES and PES. Suppose p_N and p_D be sampling probability of NES and PES respectively to achieve the same RSE. Using the result of Theorems 1 and 3, we need to have $\Delta_\sigma^{-1/2} \approx \Delta_g^{-1/2}$. Replace $\Delta_\sigma = p_D q\Delta$ and $\Delta_g = p_N^2\Delta$. Recall that q is the sampling probability of preserving candidate wedges in pool σ. Suppose the size of pool σ be the same as the size of $|g|$, i.e. $|\sigma| = |g|$. Thus, $q \approx M/\Lambda$. After some math simplifications, we get

Corollary 1. *Suppose pool size be $|\sigma| = |g| = p_D M$ in PES. The ratio between sampling probabilities of PES and NES to achieve the same RSE is given by*

$$\frac{p_N}{p_D} \approx \frac{M}{p_N\Lambda}. \tag{11}$$

Corollary 1 says that the sample size ratio between PES and NES depends on M, Λ, and sampling probability of NES (p_N). Recall that M and Λ are the number of edges and the count of wedges in the input graph.

To verify the corollary, the parameters of the methods were set to achieve the RSEs between 0.1 and 0.4. Note that we set up the size of pool as $p_D M$ in PES, i.e. $|\sigma| = |g|$. The observed and estimated ratios based on Eq. 11 are reported in Fig. 7. It can be seen that the observed ratios support our theoretical results in Eq. 11, i.e., the estimated ratios based on Eq. 11 fit the observed values very well in most of the

representative graphs. However, as expected there is a small gap between the observed and estimated ratios in a few cases.

VI. CONCLUSION AND DISCUSSION

This paper proposes a streaming algorithm called PES. It improves NES by increasing the chance of observing a triangle over a stream from p^2 in NES to pq, where q is greater than p and it is automatically adjusted over the stream. PES outperforms GPS-In consistently in all the datasets that have been tested. The performance ratio can be as high as 11. An important observation is that the performance ratio grows exponentially with data size, indicating that we could observe higher performance gain in larger datasets. We have tested on networks with 65 million nodes. Due to the prohibitive cost to calculate the ground truths (such as triangle, wedges, and shared wedges and triangles) of very large graph, we did not experiment with even larger networks. We should note that real networks often have billions of nodes, much larger than our experimented data. We expect that our algorithm would be particularly useful in such very large networks.

In retrospect, the key to improve the performance is to identify triangles as many as possible during the sampling process. In the streaming model, we need to scan each edge anyway. Thus, NES fits naturally with the streaming model because the closeness check almost comes free, especially because the sample size is small compared with the original graph. PES improves NES further by increasing the sampling
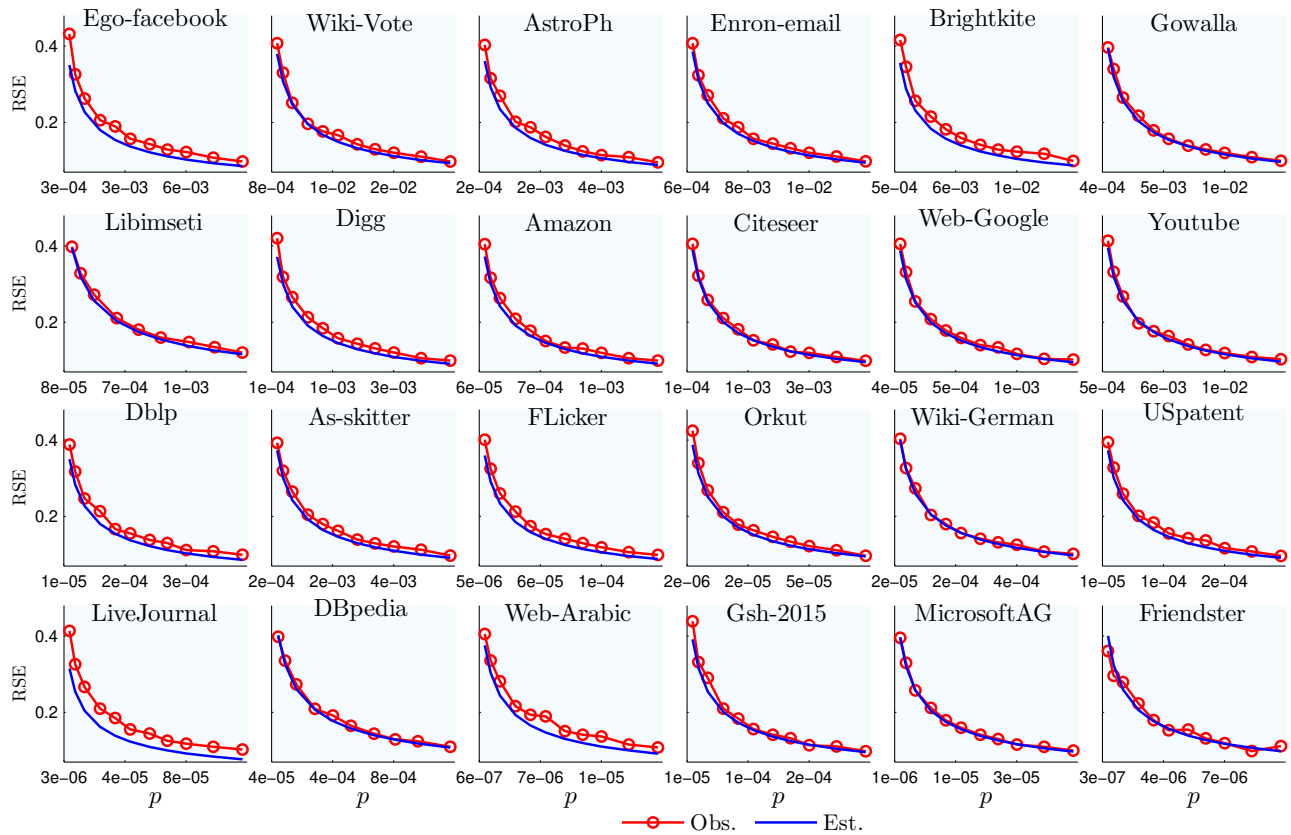
Fig. 5: The observed RSEs of $\widehat{\Delta}_{PES}$ support our estimated RSEs based on Eq. 9.
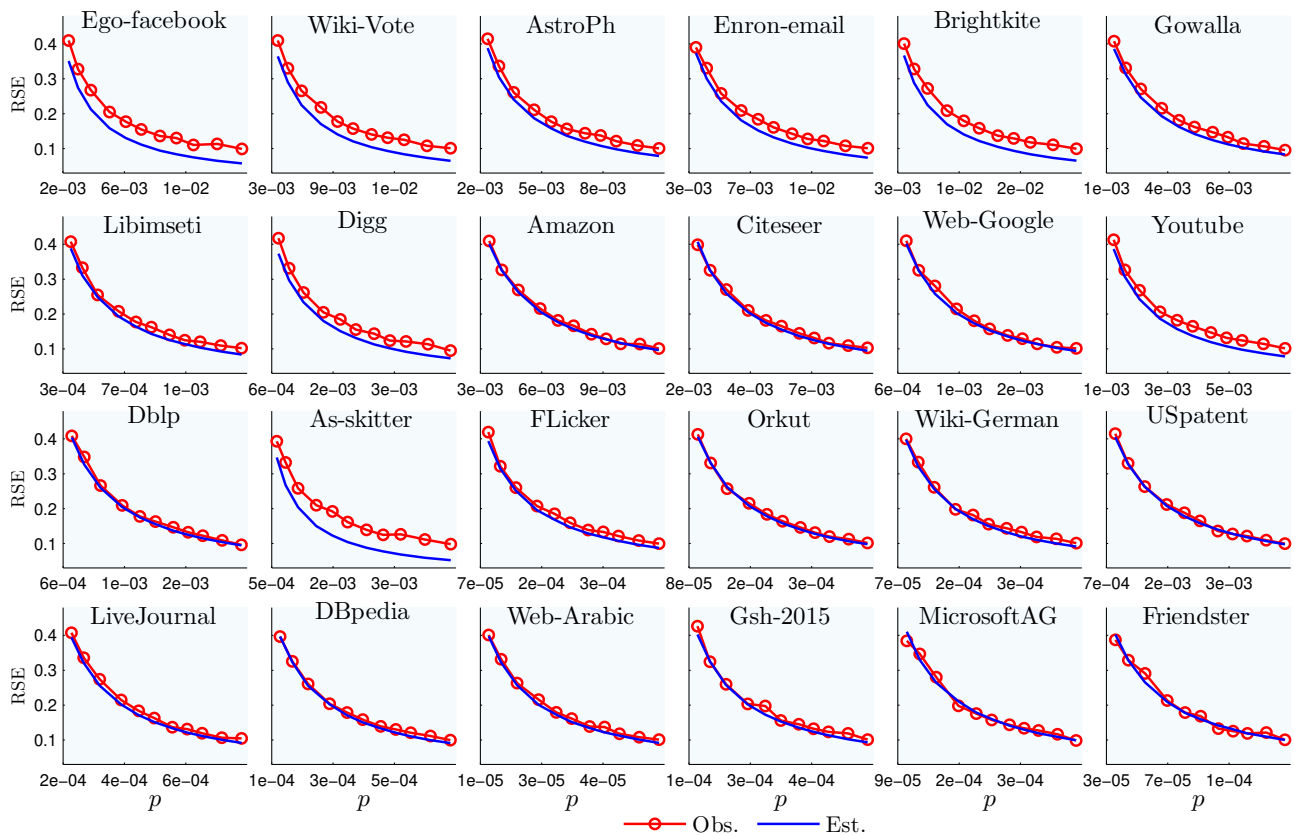

Fig. 6: The observed RSEs of $\widehat{\Delta}_{NES}$ fit very well our estimated RSEs based on Eq. 1.
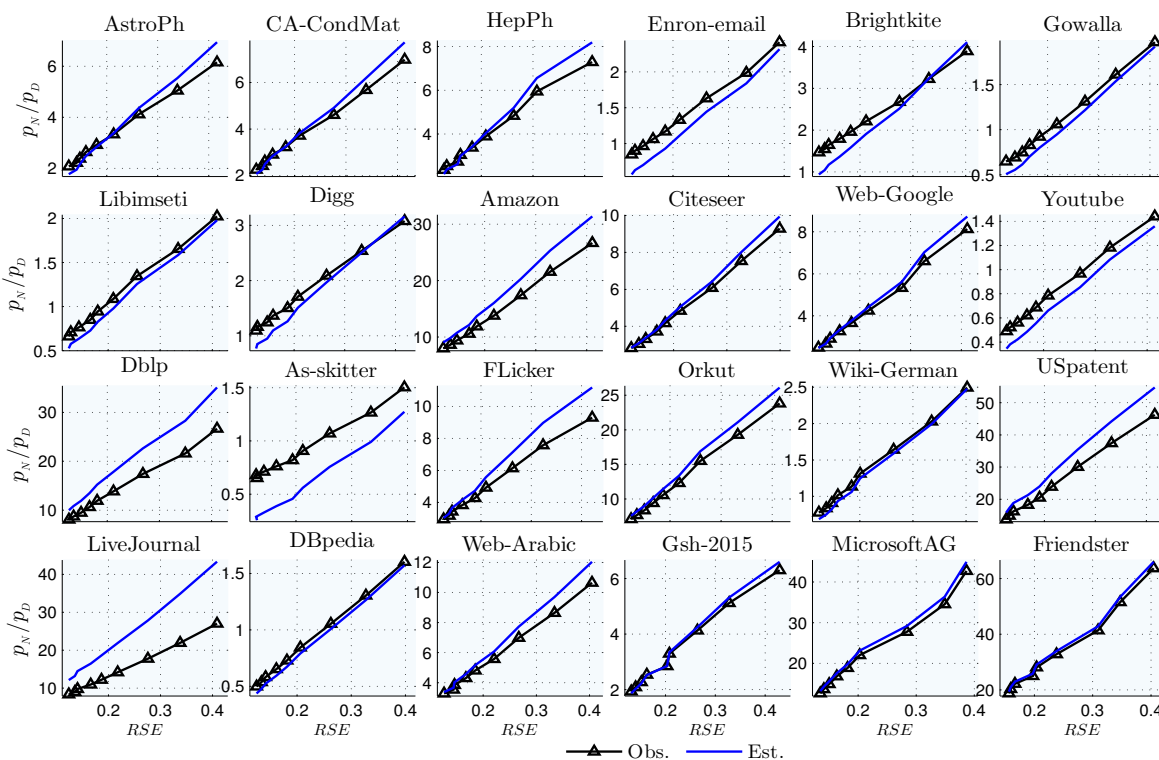
Fig. 7: Our PES outperforms NES. The observed and estimated ratios between p_N and p_D when the methods achieve the same RSEs between 0.1 and 0.4. The estimated ratios are obtained using Eq. 11.

probability of the second edge of the triangle. It improves GPS-In because GPS-In does not always add the second edge as we did in PES.

Most algorithms are compared empirically only. This is limited, and conclusions may not be true for other datasets. We compare NES and PES analytically, and quantify the performance gain. The analytical comparison also gives us a deeper understanding as for when PES is better. PES hinges on the value of q. Probability q becomes larger than p when the graph becomes larger.

VII. ACKNOWLEDGMENTS

REFERENCES

[1] H. Yin, A. R. Benson, J. Leskovec, and D. F. Gleich, "Local higher-order graph clustering," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2017, pp. 555–564.

[2] Y. Liu, C. Zhao, X. Wang, Q. Huang, X. Zhang, and D. Yi, "The degree-related clustering coefficient and its application to link prediction," *Physica A: Statistical Mechanics and its Applications*, vol. 454, pp. 24–33, 2016.

[3] P. O. Boykin and V. P. Roychowdhury, "Leveraging social networks to fight spam," *Computer*, vol. 38, no. 4, pp. 61–68, 2005.

[4] A.-L. Barabasi and Z. N. Oltvai, "Network biology: understanding the cell's functional organization," *Nature reviews genetics*, vol. 5, no. 2, p. 101, 2004.

[5] G. Kalna and D. J. Higham, "A clustering coefficient for weighted networks, with application to gene expression data," *Ai Communications*, vol. 20, no. 4, pp. 263–271, 2007.

[6] R. Goldstein and M. S. Vitevitch, "The influence of clustering coefficient on word-learning: how groups of similar sounding words facilitate acquisition," *Frontiers in psychology*, vol. 5, p. 1307, 2014.

[7] M. Latapy, "Main-memory triangle computations for very large (sparse (power-law)) graphs," *Theoretical Computer Science*, vol. 407, no. 1, pp. 458–473, 2008.

[8] R. Etemadi, J. Lu, and Y. H. Tsin, "Efficient estimation of triangles in very large graphs," in *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, ser. CIKM '16. ACM, 2016, pp. 1251–1260.

[9] 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,* 837-846. ACM, 2009.

[10] N. K. Ahmed, N. Duffield, T. L. Willke, and R. A. Rossi, "On sampling from massive graph streams," *Proceedings of the VLDB Endowment*, vol. 10, no. 11, pp. 1430–1441, 2017.

[11] 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,* 1446-1455. ACM, 2014.

[12] M. Jha, C. Seshadhri, and A. Pinar, "A space efficient streaming algorithm for triangle counting using the birthday paradox," in *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '13. New York, USA: ACM, 2013, pp. 589–597.

[13] L. De Stefani, A. Epasto, M. Riondato, and E. Upfal, "$triest$: Counting local and global triangles in fully-dynamic streams with fixed memory size," in *Proceedings of the 22th ACM KDD international conference on Knowledge discovery and data mining*. ACM, 2016.

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

[15] Y. Lim and U. Kang, "Mascot: Memory-efficient and accurate sampling for counting local triangles in graph streams," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining,* 685-694. ACM, 2015.

[16] L. S. Buriol, G. Frahling, S. Leonardi, A. Marchetti-Spaccamela, and C. Sohler, "Counting triangles in data streams," in *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems,* 253-262. ACM, 2006.

[17] Z. Bar-Yossef, R. Kumar, and D. Sivakumar, "Reductions in streaming algorithms, with an application to counting triangles in graphs," in *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms,* 623-632. Society for Industrial and Applied Mathematics, 2002.

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

[19] M. Al Hasan and V. S. Dave, "Triangle counting in large networks: a review," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 8, no. 2, p. e1226, 2018.

[20] R. Pagh and C. E. Tsourakakis, "Colorful triangle counting and a mapreduce implementation," *Information Processing Letters*, vol. 112, no. 7, pp. 277–281, 2012.

[21] M. Jha, C. Seshadhri, and A. Pinar, "A space-efficient streaming algorithm for estimating transitivity and triangle counts using the birthday paradox," *ACM Trans. Knowl. Discov. Data*, vol. 9, no. 3, pp. 15:1–15:21, Feb. 2015.

[22] L. Becchetti, P. Boldi, C. Castillo, and A. Gionis, "Efficient semi-streaming algorithms for local triangle counting in massive graphs," in *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM, 2008, pp. 16–24.

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

[24] M. Rahman and M. A. Hasan, "Sampling triples from restricted networks using mcmc strategy," in *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management,* ser. CIKM '14. New York, NY, USA: ACM, 2014, pp. 1519–1528.

[25] S. J. Hardiman and L. Katzir, "Estimating clustering coefficients and size of social networks via random walk," in *Proceedings of the 22nd international conference on World Wide Web.* ACM, 2013, pp. 539–550.

[26] R. Etemadi and J. Lu, "Bias correction in clustering coefficient estimation," in *2017 IEEE International Conference on Big Data (Big Data)*, Dec 2017, pp. 606–615.

[27] T. Schank and D. Wagner, "Approximating clustering coefficient and transitivity," *Journal of Graph Algorithms and Applications*, vol. 9, no. 2, pp. 265–275, 2005.

[28] C. Seshadhri, A. Pinar, and T. G. Kolda, "Triadic measures on graphs: The power of wedge sampling," in *SIAM International Conference on Data Mining (SDM).* SIAM, 2013, pp. 10–18.

[29] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection," http://snap.stanford.edu/data, Jun. 2014.

[30] J. Kunegis, "Konect - the koblenz network collection," http://konect.uni-koblenz.de/networks, May 2016.

[31] P. Boldi, M. Rosa, M. Santini, and S. Vigna, "Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks," in *Proceedings of the 20th international conference on WWW,* 587-596. ACM, 2011.

[32] P. Boldi and S. Vigna, "The WebGraph framework I: Compression techniques," in *Proceeding of the Thirteenth International World Wide Web Conference,* 595-601. ACM, 2004.

[33] A. Sinha, Z. Shen, Y. Song, H. Ma, D. Eide, B.-j. P. Hsu, and K. Wang, "An overview of microsoft academic service (mas) and applications," in *Proceedings of the 24th International Conference on World Wide Web.* ACM, 2015, pp. 243–246.

Appendix

A. proof of Theorem 1

Proof. Let Δ_g be the number of triangles identified by Algorithm 1 based on wedges in g. Lets δ_i be an indicator for i^{th} triangle in the original graph. δ_i is 1 when the ith triangle is identified; otherwise it is 0. Applying the variance on $\widehat{\Delta}_{NES}$ is we get:

$$var(\widehat{\Delta}_{NES}) = var\left(\frac{\Delta_g}{p^2}\right) = \frac{1}{p^4}var\left(\sum_{i=1}^{\Delta}\delta_i\right) \qquad (12)$$

$$= \frac{1}{p^4}\left(\sum_{i=1}^{\Delta}var(\delta_i) + \sum_{i\neq j}cov(\delta_i,\delta_j)\right). \quad (13)$$

The probability of identifying triangle δ_i by Algorithm 1 is p^2. Hence, the variance of δ_i is $\mathbb{E}(\delta_i)-\mathbb{E}^2(\delta_i) = p^2-p^4$. Therefore the cost of the first term in Eq. 13 is $\Delta(p^2 - p^4)$. Clearly, the covariance of two independent triangles is zero. Thus, we need to consider the covariance of dependent pairs. Each pair of shared triangles has five edges. To identify such a pair by Alg. 1, a shard edge and one of the other two edges of each triangle need to be sampled with probability p. Therefore, a chance of identifying such a pair is p^3. Obviously, the five edges of a pair of shared triangles appears in 120 different orders in a stream; in 64 out of 120 cases, such a pair can be identified. Clearly, the cost of covariance of two shared triangles is p^3-p^4. Recall that the total number of pairs of shared triangles is denoted by Φ. Therefore, the cost of covariance term is $\frac{16}{15}\Phi(p^3 - p^4)$. Thus, by adding all the costs we have

$$var(\widehat{\Delta}_{NES}) = \frac{\Delta(1-p^2)}{p^2} + \frac{16\Phi(1-p)}{15p}. \qquad (14)$$

Translate Eq. 14 into RSE , we got

$$RSE(\widehat{\Delta}_{NES}) \approx \left[\frac{1}{\Delta p^2}(1-p) + \frac{16\Phi}{15\Delta^2 p^2}(p-p^2)\right]^{1/2}.$$

When sampling probability p is small terms $-p$ and $\frac{16\Phi}{15\Delta}(p-p^2)$ are ignorable. Therefore, the RSE is $(\Delta p^2)^{-1/2}$. Replace Δp^2 by Δ_g, we obtain the theorem.