Noname manuscript No.
(will be inserted by the editor)

FURL: Fixed-memory and Uncertainty Reducing Local Triangle Counting for Multigraph Streams

**Minsoo Jung · Yongsub Lim ·
Sunmin Lee · U Kang**

Received: date / Accepted: date

Abstract Given a multigraph stream (e.g., Facebook messages or network traffic) where duplicate edges arrive continuously, how can we accurately and memory-efficiently estimate local triangles for all nodes? Local triangle counting in a graph stream is one of the fundamental tasks in graph mining with important applications including anomaly detection, social role identification, community detection, etc. Many recent graph streams include duplicate edges, hence form multigraph streams: e.g., many network packets might have a same (source, destination) pair. Although there have been several local triangle counting methods for multigraph streams, they have problems in terms of accuracy and memory efficiency; furthermore, most methods support either binary or weighted counting, and thus cannot find anomalies whose detection requires both types of counting.

In this paper, we propose FURL, a memory-efficient and accurate local triangle counting method for multigraph streams. FURL has two main advantages. First, FURL improves accuracy by 1) reducing the variance of its estimation via a regularization strategy, and 2) sampling more triangles than the state-of-the-art methods do, by using its memory space efficiently. Second, FURL finds anomalies which state-of-the-art methods cannot discover. Experimental results show that FURL outperforms state-of-the-art methods in terms of accuracy and memory efficiency. Thanks to FURL, we discover interesting anomalies from a Bitcoin network.

Minsoo Jung
Seoul National University
minsoojung@snu.ac.kr

Yongsub Lim
MakinaRocks
yongsub@makinarocks.ai

Sunmin Lee
Seoul National University
smileeesun@snu.ac.kr

U Kang
Seoul National University
ukang@snu.ac.kr (corresponding author)

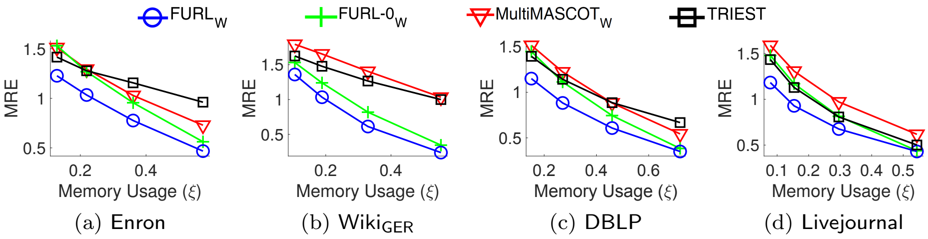


Fig. 1: Mean of relative error (MRE) vs. memory usage of our proposed FURL compared to competing methods $\text{MultiMascot}_{\text{W}}$ and TRIEST for weighted local triangle counting in a multigraph stream. Our best proposed method FURL_{W} outperforms the other methods. MRE of FURL_{W} is $1.23\times \sim 4.33\times$ and $1.15\times \sim 4.19\times$ smaller than that of $\text{MultiMascot}_{\text{W}}$ and TRIEST, respectively. Also, MRE of FURL_{W} is up to $1.45\times$ smaller than that of FURL-0_{W} in almost all cases since FURL_{W} gives concentrated results.

Keywords Local triangle counting · Graph stream · Edge sampling

1 Introduction

How can we accurately estimate local triangles for all nodes with a fixed memory size in a multigraph stream? The *local triangle counting* problem is to count the number of triangles containing each node in a graph and has been extensively studied because of its wide and important applications. For instance, it has been used for social role identification of a user (Welser et al., 2007; Chou and Suzuki, 2010), content quality evaluation (Becchetti et al., 2010), data-driven anomaly detection (Becchetti et al., 2010; Yang et al., 2011; Lim and Kang, 2015; Lim et al., 2018), community detection (Berry et al., 2011; Suri and Vassilvitskii, 2011), motif detection (Milo et al., 2002), clustering ego-networks (Epasto et al., 2015), and uncovering hidden thematic layers (Eckmann and Moses, 2002). Also, recent real-world graph streams contain duplicate edges, i.e., they are multigraph streams. Examples include a communication network in Internet, a phone call history, SNS messages like tweets, etc. In such environments, it is crucial to carefully handle duplicate edges in local triangle counting (Lim et al., 2018; Stefani et al., 2017; Jha et al., 2015; Wang et al., 2017).

Although a number of methods have been successfully applied to local triangle counting in a multigraph stream, existing methods still have three challenging issues. First, they have a large variance which causes low accuracy. In a graph stream model where edges continuously arrive, only one trial is allowed and a large variance causes a large difference between estimated and true local triangle counts. They do not produce stable results and show bad worst case performance due to their large variance. Second, the state-of-the-art methods (Stefani et al., 2017; Wang et al., 2017) support only either binary or weighted counting which are two representative approaches to count triangles in a multigraph stream. Thus, they cannot find anomalous nodes whose detection requires both types of triangle counting, as described in Section 4.4. Third, the existing methods have limitations: the amount of required memory space depends on stream length (Lim et al., 2018), memory space is unnecessarily used for handling duplicate edges (Stefani et al., 2017) or memory space is not fully used (Wang et al., 2017).

In this paper, we propose FURL (**F**ixed-memory and **U**ncertainty **R**educing **L**ocal Triangle Counting for Multigraph Streams), a novel local triangle counting algorithm for a multigraph stream. FURL is based on edge sampling from the

Table 1: Comparison of FURL and other algorithms. Our main proposed method FURL shows the best performance for all metrics.

	Proposed (main)	Proposed (Basic)	Existing		
	FURL	FURL-0	TRIEST Stefani et al. (2017)	PartitionCT Wang et al. (2017)	MultiMascot Lim et al. (2018)
Error	**Small**	Medium	Large	Large	Large
Memory Efficiency	**High**	High	Low	Low	Low
Binary Counting	✓	✓		✓	✓
Weighted Counting	✓	✓	✓		✓

multigraph stream, and uses only a fixed number of edges regardless of the input size which can possibly be infinite. Before describing FURL, we first present a basic method FURL-0. FURL-0 applies reservoir sampling with a random hash, which samples a fixed number of distinct items in a stream uniformly at random, for handling duplicate edges. Our main proposed method FURL further improves on FURL-0 by a regularization strategy of combining the current estimation with past estimations to decrease variance and improve accuracy.

Compared to previous methods (Lim et al., 2018; Stefani et al., 2017; Wang et al., 2017), FURL has three main advantages: accuracy, memory-efficiency, and supporting both binary and weighted counting, as summarized in Table 1. First, FURL provides the best accuracy by decreasing the variance of the estimation (i.e., increasing the probability that the estimation is close to the truth) via a regularization strategy based on ensembles. Low variance estimation is especially useful in a graph stream setting where each element can be seen only once. Second, FURL stores only a fixed number of edges to count the number of triangles, regardless of an input graph stream size. This characteristic enables FURL to 1) analyze a graph stream of any size, unlike a previous method (Lim et al., 2018) whose memory usage is proportional to the graph size and thus eventually runs out of memory, and 2) provide better accuracy than the competitors (Lim et al., 2018; Wang et al., 2017) under the same memory usage since FURL samples more triangles by fully using the memory space. Also, FURL efficiently uses memory in handling duplicate edges compared to the state-of-the-art method (Stefani et al., 2017) which allocates memories unnecessarily for the same duplicate edges. Third, FURL supports both ways of counting triangles: binary and weighted counting. Thus, FURL finds anomalous nodes the state-of-the-art methods (Stefani et al., 2017; Wang et al., 2017) cannot find (Section 4.4).

Our contributions are summarized as follows.

- **Algorithm.** We propose FURL, an accurate and memory-efficient local triangle counting algorithm for a multigraph stream. FURL improves the accuracy of previous multigraph stream algorithms by decreasing variance. FURL uses a fixed amount of edges to count the number of triangles, and thus it can handle streams of any sizes. Furthermore, FURL supports both of the representative ways of triangle counting in a multigraph stream: binary and weighted counting.

Table 2: Table of symbols.

Symbol	Description
S	Graph stream
V, E	Set of nodes and edges
u, v, w	Nodes
e	Edges
N_u	Set of neighbors of u
N_{uv}	Set of common neighbors of u and v
n, m	Number of nodes and edges
o_e	Number of occurrence of an edge e in a multigraph stream
$u(T)$	Number of unique edges in a stream at time T
D	Buffer
$\|D\|$	Number of elements stored in buffer D
M	Buffer size
h_{max}	Maximum hash value in a buffer
D_{max}	Edge whose hash value is h_{max}
b	Current bucket
b_λ	Bucket that λ appears
J	Bucket size
δ	Decaying factor for past estimations
λ	Triangle
T, t	Current time, and time
T_λ	Time that the last edge of a triangle λ arrives
T_M	The last time when local triangle estimations equal the true triangle counts
Δ_u	True local triangle count of node u
c_u	Estimated local triangle count of node u in FURL-0
τ_u	Estimated local triangle count of node u in FURL
q_λ	Triangle estimation weight of a triangle λ
X_λ	Estimated triangle counts by FURL-0
Y_λ	Estimated triangle counts by FURL

– **Analysis.** We give a full theoretical analysis of FURL, including expectation and variance of our estimations, and provable guarantees that our method gives superior accuracy.
– **Performance.** We demonstrate that FURL outperforms the existing methods in terms of accuracy and memory-efficiency (Figure 1).
– **Discovery.** We show that FURL discovers anomalies using both weighted and binary triangle counts, which state-of-the-art methods cannot discover. Applying FURL to real-world Bitcoin network data, we present interesting anomalies, a gambling site and a Bitcoin mining pool.

The code and datasets used in the paper are available in `http://datalab.snu.ac.kr/furl`. The rest of this paper is organized as follows. Section 2 gives related works and preliminaries. Section 3 describes our proposed method FURL. In Section 4, we evaluate the performance of FURL and give discovery results. We conclude in Section 5. Table 2 shows the symbols used in this paper.

2 Related Works

We present related works about local triangle counting and reservoir sampling.

2.1 Local Triangle Counting

A simple and fast algorithm for local triangle counting is to get A^3 for an adjacency matrix A (Alon et al., 1997). It takes only $O(m^{1.41})$ time, but its memory usage, $O(n^2)$, is quadratic. Latapy (Latapy, 2008) presented a fast and space-efficient algorithm, but in a non-stream setting. To handle simple graph streams, Becchetti et al. (Becchetti et al., 2010) devised a semi-streaming algorithm based on min-wise independent permutations (Broder et al., 2000). Their algorithm requires $O(n)$ space in main memory and $O(\log(n))$ sequential scans over the edges of the graph. However, it is inappropriate for handling a graph stream in real time due to its multiple scans of the graph. Kutzkov and Pagh (Kutzkov and Pagh, 2013) proposed a randomized one pass algorithm based on node coloring (Pagh and Tsourakakis, 2012). Despite the property of scanning once, it is limited in practice because it requires prior knowledge about the graph. These problems are resolved by MASCOT (Lim and Kang, 2015) which is based on a triangle counting with sampling (Tsourakakis et al., 2009). MASCOT takes only one parameter of edge sampling probability with one sequential scan of the graph stream and estimates the number of triangles based on the sampled graph which consists of edges sampled so far. But MASCOT endlessly samples edges with a static edge sampling probability and then eventually the out-of-memory error will happen. De Stefani et al. (Stefani et al., 2016) proposed TRIEST which computes the number of triangles in fully-dynamic streams with a fixed number of sampled edges using reservoir sampling instead static sampling probability, but it does not consider duplicated edges. An extended version MULTIMASCOT of MASCOT for a multigraph stream is proposed in (Lim et al., 2018); however, their memory usage is proportional to stream length. De Stefani et al. (Stefani et al., 2017) also extended TRIEST for a multigraph stream but the extended algorithm wastes memory to handle duplicate edges since it deals with duplicate edges as different ones. Wang et al. (Wang et al., 2017) proposed PARTITIONCT based on k partition sketch (Flajolet and Martin, 1985). It computes the number of triangles in a multigraph stream using a fixed number of buckets, but it does not use all the buckets since it randomly decides which bucket stores an edge. Additionally, TRIEST and PARTITIONCT support only either binary or weighted triangle counting. In these existing methods, large variances in the outputs cause low accuracy since accuracy heavily depends on a variance in a stream setting. Our proposed algorithm FURL improves the accuracy by reducing the variance.

2.2 Reservoir Sampling

Reservoir sampling is a technique to sample a given number of elements from a stream uniformly at random (Vitter, 1985). Let D be an array buffer of size M. For each item e arriving at time $T \geq 1$, the sampling procedure is as follows. If $T \leq M$, e is unconditionally stored in D_T, the Tth slot of D. If $T > M$, first pick a random integer i from $\{1, \ldots, T\}$. If $i \leq M$, the existing item at D_i is dropped and e is stored at D_i; otherwise, e is discarded. As a result, the sampling probability for every observed item becomes $\min(M/T, 1)$ at time T.

Reservoir Sampling With Random Hash. Reservoir sampling with random hash (random sort (Sunter, 1977)) is a modification of reservoir sampling to sample distinct items uniformly at random in a stream environment. The original reservoir sampling cannot handle duplicate items since it does not know an item has been sampled or not in the past if the item is not in the buffer. The main idea is to assign each item a random hash value and to keep M items with minimum hash values. The arriving item e is sampled if $h(e) < h_{max} = \max_{f \in D} h(f)$. Then, the reservoir sampling with random hash determines whether an item has been sampled or not by comparing its hash value with the Mth minimum value. Note that duplicate items have the same hash value. The method samples distinct items uniformly at random, because it is equivalent to picking M distinct items with the minimum hash values in the whole stream. The reservoir sampling with random hash is used in our proposed FURL algorithm to process multigraph streams.

3 Proposed Method

How can we accurately estimate local triangles in a multigraph stream with a fixed memory space? In this section, we propose FURL, an accurate and memory-efficient local triangle counting algorithm for a multigraph stream to answer the question. We first present a basic algorithm FURL-0 (Section 3.1) which handles duplicate edges and uses only a fixed amount of sampled edges for triangle counting through reservoir sampling with random hash. Our main proposed algorithm FURL (Section 3.2) further improves the accuracy of FURL-0 by additionally assembling intermediate estimation results. Note that all the proposed methods store edges in a buffer D which has a fixed capacity M.

3.1 FURL-0: Basic Algorithm

There are two ways of counting duplicate edges: binary counting and weighted counting. The binary counting considers only the existence of an edge, leaving out the number of occurrences of an edge. In contrast, the weighted counting takes the number of occurrences of an edge into account. For instance, given a triangle $\lambda = (e_a, e_b, e_c)$ and a multigraph stream $(e_a, e_a, e_a, e_b, e_b, e_c)$, binary counting counts λ as 1 while weighted counting counts λ as $3 \times 2 \times 1 = 6$.

Each way of triangle counting has its own use. In binary counting, its result on a multigraph is the same as that on the corresponding simple graph. Thus, binary counting is applied to most of the existing applications where the existence or nonexistence of an edge is important, such as computing clustering coefficient. Unlike binary counting, weighted counting considers one more information, the number of duplicate edges. Weighted counting is exploited when the number of duplicate edges is an important indicator of the strength of connections: e.g., a phone call history, SNS messages, and email network.

In the following, we describe FURL-0_{B} for binary counting and FURL-0_{W} for weighted counting in a multigraph stream.

*Binary Counting (*FURL-0_{B}*).* For binary triangle counting, we sample distinct edges with an equal probability regardless of the degree of duplications: i.e., M edges in buffer D are chosen uniformly at random from a set of distinct edges observed so far.

The algorithm performs the following four steps for every new edge e: 1) add a new node to sampled graph G, 2) check if the new edge e is already sampled in the buffer, 3) if not already sampled, apply the sampling procedure to the new edge, and 4) if sampled, update local triangle estimation for the current sampled graph. The pseudo code of FURL-0_{B} is shown in Algorithm 1.

Let T be the current time and $e = (u, v)$ be a new edge arriving from a stream at time T. The first step is to add each node u and v of e to the sampled graph G and create its triangle counter if the node has not arrived before. Note that it is inevitable to use $O(n)$ memory, where n is the number of nodes, to keep counters since our method is a local triangle counting method. The second step is to examine if the buffer D contains the new edge e (line 7). If D contains e, FURL-0_{B} ignores e since FURL-0_{B} considers only the existence of an edge. If not, FURL-0_{B} continues on the third step. The third step is to sample e using the reservoir sampling with random hash (lines 15–21). In the reservoir sampling with random hash, each arriving edge e is assigned a hash value $h(e)$, and the buffer

D maintains M edges with minimum hash values. Without loss of generality, we assume the codomain of $h(e)$ is (0,1). To maintain M edges with minimum hash values, we use a priority queue (max heap) as buffer D. Then, the sampling procedure is as follows: e is sampled unconditionally until D is not full. If the buffer D cannot store an edge anymore (the buffer overflows), FURL-0_{B} discards one edge among $M+1$ edges since the buffer has only M space. To discard one edge, FURL-0_{B} compares $h(e)$ and h_{max}, where $h_{max} = \max_{f \in D} h(f)$. If $h(e) < h_{max}$, the edge with the maximum hash value in D is discarded and e is sampled. Then, the fourth step, updating local triangle counts (lines 29–35), is performed because the sampled graph G is updated by the new edge. If not, e is discarded and FURL-0_{B} skips the fourth step and proceeds to the next edge.

Let N_u and N_{uv} be a set of neighbors of u, and a set of common neighbors of u and v, respectively. Before updating the local triangle estimation, we calculate $N_{uv} = N_u \cap N_v$ for the newly sampled edge $e = (u, v)$. Each triple (w, u, v) for every $w \in N_{uv}$ is a triangle appearing on G, i.e., u and v get $|N_{uv}|$ triangles, and w gets one triangle. If D does not overflow, the triangle estimation is updated by a factor 1: i.e., a counted triangle adds 1 to our estimation. The local triangle estimation c is updated as follows: $c_z = c_z + |N_{uv}|$ for $z \in \{u, v\}$, and $c_w = c_w + 1$ for $w \in N_{uv}$. FURL-0_{B} provides the exact local triangle counts and $ExactCnt$ is $true$ during this time. If D overflows, triangle estimation is updated by a factor $q_T = \frac{M-3}{M} \cdot \frac{1}{h_{max}^3}$. Intuitively, this means we add more triangle counts to our estimation if the probability of sampling such triangle is small (i.e., h_{max} is small). Such adjustment is required to make the estimation by FURL-0_{B} unbiased as shown in Theorem 1. Note that $ExactCnt$ turns to $false$ when D overflows (line 18) since we are doing approximate counting in this case.

Theorem 1 *Let Δ_u be the true local triangle count for a node u, and c_u be the estimation given by* FURL-0_{B}*. For every node u, $\mathbb{E}[c_u] = \Delta_u$.*

Proof See Appendix.

*Weighted Counting (*FURL-0_{W}*).* In weighted triangle counting, weights of edges are multiplied in computing the number of triangles. We propose FURL-0_{W} (Algorithm 2) for weighted triangle counting in a multigraph stream. To reflect the weight of edges, i.e., the duplicate number of edges, FURL-0_{W} keeps the occurrence number o_e for an edge e in buffer D. Note that $O(M)$ memory is needed to keep the occurrence numbers.

Unlike FURL-0_{B}, FURL-0_{W} first updates local triangle estimation (lines 8–14) and then goes through the sampling procedure (lines 23–26). Also, FURL-0_{W} increases o_e by 1 if an edge e is already in the buffer. The triangle estimation is increased by a factor 1 when $ExactCnt$ is $true$ (i.e., sampling did not happen since the buffer was not full yet), and by $q_T = \frac{M-2}{M} \cdot \frac{1}{h_{max}^2}$ otherwise.

The procedure of updating triangle estimation (lines 15–22) is a bit different from that of FURL-0_{B} because weighted triangle counting considers edges' weights. When a new edge $e = (u, v)$ arrives, for each $w \in N_{uv}$, $o_{(u,w)} \cdot o_{(v,w)}$ number of triangles are created. Therefore triangle estimations c_u, c_v, and c_w are incremented by multiplying a given factor with $o_{(u,w)}$ and $o_{(v,w)}$.

FURL-0_{W} gives an unbiased estimation as shown in Theorem 2.

Algorithm 1: FURL-0_{B} for multigraph stream (binary)

Input: Graph stream S, maximum number M of edges stored, and a random hash function h
Output: Local triangle estimation c

$G \leftarrow (V, D)$ with $V = D = \emptyset$.
The estimated count c for nodes is initialized to $\emptyset$.
The boolean flag $ExactCnt$ is initialized to $true$.
foreach *edge* $e = (u, v)$ *from* S **do** `// at time` T
  $DiscoverNode(u)$.
  $DiscoverNode(v)$.
  if $e \notin D$ **then**
    $SampleNewEdge$-$\mathtt{M}_B(e)$.
    $UpdateTriangles$-$\mathtt{M}_B(e)$.
  end
end

Subroutine $DiscoverNode(u)$
  if $u \notin V$ **then** $V \leftarrow V \cup \{u\}$ and $c_u = 0$.
end

Subroutine $SampleNewEdge$-$\mathtt{M}_B(e)$
  if $|D| < M$ **then** Append e to D.
  else
    if $ExactCnt = true$ **then** $ExactCnt \leftarrow false$.
    $ReplaceEdge$-$\mathtt{M}(e)$.
  end
end

Subroutine $ReplaceEdge$-$\mathtt{M}(e)$
  if $h(e) < h_{max}$ **then**
    $D_{max} \leftarrow e'$ *such that* $h(e') = h_{max}$.
    Remove D_{max} from D.
    Append e to D.
  end
end

Subroutine $UpdateTriangles$-$\mathtt{M}_B(e)$
  if $ExactCnt = true$ **then** $IncreaseEstimation(e, 1)$.
  else if e *is sampled* **then**
    $q_T \leftarrow \frac{M-3}{M} \frac{1}{h_{max}^3}$.
    $IncreaseEstimation(e, q_T)$.
  end
end

Subroutine $IncreaseEstimation(e = (u, v),\ \theta)$
  $N_{uv} \leftarrow N_u \cap N_v$.
  foreach $w \in N_{uv}$ **do** $c_w \leftarrow c_w + \theta$.
  $c_u \leftarrow c_u + (\theta \times |N_{uv}|)$.
  $c_v \leftarrow c_v + (\theta \times |N_{uv}|)$.
end

Theorem 2 *Let Δ_u be the true local triangle count for a node u, and c_u be the estimation given by* FURL-0_{W}. *For every node u, $\mathbb{E}[c_u] = \Delta_u$.*

Proof See Appendix.

3.2 FURL: Main Algorithm

The unbiased estimation of FURL-0 guarantees the accuracy if the estimation is obtained by averaging results from multiple independent trials. In real-world

Algorithm 2: FURL-0_{W} for multigraph stream (weighted)

Input: Graph stream S, maximum number M of edges stored, and a random hash function h
Output: Local triangle estimation c

Initialize variables as in lines 1-3 of Algorithm 1.
foreach *edge* $e = (u, v)$ *from* S **do** `// at time` T
  $DiscoverNode(u)$.
  $DiscoverNode(v)$.
  $UpdateTriangles$-$\mathtt{M}_W(e)$.
  $SampleNewEdge$-$\mathtt{M}_W(e)$.
end

Subroutine $UpdateTriangles$-$\mathtt{M}_W(e)$
  if $ExactCnt$ **then** $IncreaseEstimation$-$\mathtt{M}_W(e, 1)$.
  else
    $q_T \leftarrow \frac{M-2}{M} \frac{1}{h_{max}^2}$.
    $IncreaseEstimation$-$\mathtt{M}_W(e, q_T)$.
  end
end

Subroutine $IncreaseEstimation$-$\mathtt{M}_W(e = (u, v), \theta)$
  $N_{uv} \leftarrow N_u \cap N_v$.
  foreach $w \in N_{uv}$ **do**
    $c_w \leftarrow c_w + \theta \cdot o_{(u,w)} \cdot o_{(v,w)}$.
    $c_u \leftarrow c_u + \theta \cdot o_{(u,w)} \cdot o_{(v,w)}$.
    $c_v \leftarrow c_v + \theta \cdot o_{(u,w)} \cdot o_{(v,w)}$.
  end
end

Subroutine $SampleNewEdge$-$\mathtt{M}_W(e)$
  if $e \in D$ **then** $o_e \leftarrow o_e + 1$.
  else The same as $SampleNewEdge$-$\mathtt{M}_B$.
end

scenarios of processing a graph stream, however, it requires very high costs to keep multiple independent sample graphs or scan a graph stream multiple times. It means that only one trial is allowed. Thus, the accuracy greatly depends on the variance as well as the unbiasedness. In this section, we propose FURL which has a lower variance than FURL-0 with a slightly biased estimation. The idea is to regularize the estimation such that the overall estimation error of FURL becomes smaller than that of FURL-0.

*Binary Counting (*FURL$_{\mathrm{B}}$*).* We propose FURL$_{\mathrm{B}}$ which is an improved algorithm from FURL-0_{B} for binary triangle counting. The main idea of FURL$_{\mathrm{B}}$ (Algorithm 3) is to build an ensemble estimator efficiently by combining the current estimation with estimations at earlier timesteps through a weighted average scheme. The procedure of FURL$_{\mathrm{B}}$ is very similar to that of FURL-0_{B}, but its estimation is computed by a weighted average of estimations obtained at previous times.

Let T_M be the last time when $c^{(t)}$ equals the true triangle counts and T be the current time. We first divide the interval $[T_M + 1, T]$ into buckets of size $J > 0$. We consider $[1, T_M]$ as a bucket 0. Let $c^{(t)}$ and $\tau^{(t)}$ be the estimations of FURL-0_{B} and FURL$_{\mathrm{B}}$ at time t respectively. For $T \leq T_M$, $\tau^{(t)} = c^{(t)}$, i.e., when the bucket does not overflow, the estimation of FURL$_{\mathrm{B}}$ is the same as that of FURL-0_{B}. Note that T_M is set to $T - 1$ (line 31) where T is the time that the buffer D first overflows, since after then the counting becomes approximate. For time

$T > T_M$, at the boundary of each bucket $i > 0$, FURL_B updates the estimation by $\tau^{(T_M+iJ)} = \delta\tau^{(T_M+(i-1)J)} + (1-\delta)c^{(T)}$ where $T = T_M + iJ$ (lines 14–18). Let us consider the boundary of a bucket b at time T; the past estimation $c^{(T_M+iJ)}$ has been weighted by $W(i)$ where $W(i) = (1-\delta)\delta^{b-i}$ for $i \geq 1$ and δ^b for $i = 0$. This is because for $i \geq 1$, $c^{(T_M+iJ)}$ is weighted by $(1-\delta)$ at time $T_M + iJ$ and additionally weighted by δ at each successive bucket boundary; for $i = 0$, $c^{(T_M+iJ)} = c^{(T_M)}$ is weighted by δ at the boundary of every bucket $j \geq 1$.

In Algorithm 3, FURL_B internally uses FURL-0_B and accepts two more parameters: a bucket size J and a decaying factor $0 \leq \delta < 1$. Note that FURL_B with $\delta = 0$ is the same as FURL-0_B. The subroutine *Query* (lines 19–26) is designed to process a query given at any time T. If $T \leq T_M$, the current c is returned; if $T > T_M$ is a bucket boundary, i.e., $(T - T_M) \bmod J = 0$, the current $\hat{\tau}$ is returned because $\hat{\tau}$ is already up-to-date by line 16; if $T > T_M$ is not a bucket boundary, FURL_B returns $\delta\hat{\tau} + (1-\delta)c$. Note that we use a condition $\mathit{ExactCnt} = \mathit{true}$ to check whether we have the exact count or not (line 21).

We prove FURL_B gives more accurate results, i.e., closer to the ground truth, than FURL-0_B in Theorem 3. We first state Lemmas 1 and 2 used in Theorem 3.

Lemma 1 *Let $b_\lambda > 0$ be the bucket where a triangle λ is formed, and Y_λ be the estimated count of a triangle λ by* FURL_B*. Then,*

$$\mathbb{E}[Y_\lambda] = 1 - \phi(b_\lambda),$$

where $\phi(i) = \delta^{b-i+1}$ and b is the current bucket.

Proof See Appendix.

Lemma 2 *Let $b_\lambda > 0$ be the bucket where a triangle λ is formed, and Y_λ be the estimated count of a triangle λ by* FURL_B*. Then,*

$$\mathrm{Var}[Y_\lambda] = (1 - \phi(b_\lambda))^2 (k_{T_\lambda} - 1),$$

where T_λ is the first time all three edges of λ arrive, $k_{T_\lambda} = \frac{(M-3)(u(T_\lambda)-3)(u(T_\lambda)-4)(u(T_\lambda)-5)}{M(M-4)(M-5)(M-6)}$, $\phi(i) = \delta^{b-i+1}$, and b is the current bucket.

Proof See Appendix.

The following theorem shows FURL_B is more accurate than FURL-0_B, giving concentrated results.

Theorem 3 *Let Y_λ and X_λ be the estimated counts of a triangle λ by* FURL_B *and* FURL-0_B*, respectively. Consider any triangle λ that is counted at time $T_\lambda > T_M$. Let $u(T)$ be the number of unique edges that have arrived at time T. If $u(T_\lambda) \geq \sqrt[3]{\frac{1+\alpha}{\alpha}}M + 3$, the interval by $\mathbb{E}[Y_\lambda] \pm \alpha \cdot \mathrm{Var}[Y_\lambda]$ is strictly included in that by $\mathbb{E}[X_\lambda] \pm \alpha \cdot \mathrm{Var}[X_\lambda]$ for any α.*

Proof See Appendix.

Algorithm 3: $\mathrm{FURL_B}$ for multigraph stream (binary)

Input: Graph stream S, maximum number M of edges stored, interval J for updating estimation, and decaying factor δ for past estimation
Output: Local triangle estimation $\tau = Query(\delta)$

```
Initialize variables as in lines 1-3 of Algorithm 1.
The averaged estimation τ for nodes is initialized to ∅.
The time T and T_M are initialized to 0.
foreach edge e = (u, v) from S do  // at time T
    T ← T + 1.
    DiscoverNode(u).
    DiscoverNode(v).
    WeightedAverage(δ).
    if e ∉ D then
        SampleNewEdge-MX_B(e).
        UpdateTriangles-M_B(e). // c is computed here.
    end
end
Subroutine WeightedAverage(δ)
    if ExactCnt = false and (T − T_M) mod J = 0 then
        τ̂ ← δτ̂ + (1 − δ)c.
    end
end
Subroutine Query(δ)
    // Exact counting; T_M is not set yet.
    if ExactCnt = true then return c.
    // Approximate counting; T_M is set.
    else if T ≤ T_M then return c.
    else if (T − T_M) mod J = 0 then return τ̂.
    else return δτ̂ + (1 − δ)c.
end
Subroutine SampleNewEdge-MX_B(e)
    if |D| < M then Append e to D.
    else
        if ExactCnt = true then
            T_M ← T − 1.
            τ̂ ← c. // The exact counts at T_M are saved here.
            ExactCnt ← false.
        end
        ReplaceEdge-M(e).
    end
end
```

Weighted Counting ($\mathrm{FURL_W}$*).* We propose $\mathrm{FURL_W}$ which improves the weighted local triangle counting algorithm FURL-0_W. $\mathrm{FURL_W}$ (Algorithm 4) first goes through updating triangle estimation and sampling process as in FURL-0_W, and then combines past estimations with the current one.

$\mathrm{FURL_W}$ gives more accurate results than FURL-0_W, as described in Theorem 4. We first state Lemmas 3 and 4 used in Theorem 4.

Lemma 3 *Let b_λ be the bucket where λ is formed and Y_λ be the estimated count of a triangle λ with $b_\lambda > 0$ by* $\mathrm{FURL_W}$*. For every triangle λ,*

$$\mathbb{E}\left[Y_\lambda\right] = 1 - \phi(b_\lambda),$$

where $\phi(i) = \delta^{b-i+1}$ and b is the current bucket.

Proof See Appendix.

Lemma 4 *Let b_λ be the bucket where λ is formed and Y_λ be the estimated count of a triangle λ with $b_\lambda > 0$ by* $\mathrm{FURL_W}$*. For every triangle λ,*

$$\mathrm{Var}\left[Y_\lambda\right] = \left(1 - \phi(b_\lambda)\right)^2 \left(l_{T_\lambda} - 1\right),$$

where T_λ is the first time all three edges of λ arrive,
$l_{T_\lambda} = \frac{(M-2)(u(T_\lambda-1)-2)(u(T_\lambda-1)-3)}{M(M-3)(M-4)}$, $\phi(i) = \delta^{b-i+1}$, and b is the current bucket.

Proof See Appendix.

The following theorem shows $\mathrm{FURL_W}$ is more accurate than FURL-0_W, giving concentrated results.

Theorem 4 *Let Y_λ and X_λ be the estimated counts of a triangle λ by* $\mathrm{FURL_W}$ *and* FURL-0_W*, respectively. Consider any triangle λ that is counted at time $T_\lambda > T_M$. If $u(T_\lambda - 1) \geq \sqrt{\frac{1+\alpha}{\alpha}}M + 2$, the interval by $\mathbb{E}\left[Y_\lambda\right] \pm \alpha \cdot \mathrm{Var}\left[Y_\lambda\right]$ is strictly included in that by $\mathbb{E}\left[X_\lambda\right] \pm \alpha \cdot \mathrm{Var}\left[X_\lambda\right]$ for any α.*

Proof See Appendix.

Additionally, we prove the interval $\mathbb{E}\left[Y_\lambda\right] \pm \alpha \cdot \mathrm{Var}\left[Y_\lambda\right]$ of FURL is included in the interval $\mathbb{E}\left[X_\lambda\right] \pm \alpha \cdot \mathrm{Var}\left[X_\lambda\right]$ of FURL-0 in terms of estimated counts for a node, as described in Theorem 5. We first state Lemmas 5 and 6 used in Theorem 5.

Lemma 5 *Let Δ_u be the true local triangle count for a node u and Y_u be the estimation given by* FURL *for node u. Then,*

$$(1 - \delta)\,\Delta_u < \mathbb{E}\left[Y_u\right] \leq \Delta_u.$$

Proof See Appendix.

Lemma 6 *Let X_u and Y_u be the estimated local triangle count for a node u given by* FURL-0 *and* FURL *respectively, Λ_u be the set of triangles containing node u, and b_λ be the bucket where a triangle λ is formed. Then,*

$$(1 - \delta)^2\,\mathrm{Var}\left[X_u\right] \leq \mathrm{Var}\left[Y_u\right] \leq \left(1 - \delta^b\right)^2 \mathrm{Var}\left[X_u\right]$$

where b is the current bucket.

Proof See Appendix.

Theorem 5 *Let $u(T)$ be the number of unique edges that have arrived at time T. If $\delta^b > 1 - \sqrt{1 - \frac{\mathbb{E}[X_u]}{\alpha \mathrm{Var}[X_u]}}$, the interval by $\mathbb{E}\left[Y_\lambda\right] \pm \alpha \cdot \mathrm{Var}\left[Y_\lambda\right]$ is strictly included in that by $\mathbb{E}\left[X_\lambda\right] \pm \alpha \cdot \mathrm{Var}\left[X_\lambda\right]$ for any α.*

Proof See Appendix.

Algorithm 4: FURL_W for multigraph stream (weighted)

Input: Graph stream S, maximum number M of edges stored, interval J for updating estimation, and decaying factor δ for past estimation
Output: Local triangle estimation $\tau = Query(\delta)$

```
Initialize variables as in lines 1-3 of Algorithm 3.
foreach edge e = (u, v) from S do  // at time T
    T ← T + 1.
    DiscoverNode(u).
    DiscoverNode(v).
    WeightedAverage(δ).
    UpdateTriangles-M_W(e). // c is computed here.
    SampleNewEdge-MX_W(e).
end

Subroutine SampleNewEdge-MX_W(e)
    if e ∈ D then  O_e ← O_e + 1.
    else if |D| < M then  Append e to D.
    else
        if ExactCnt = true then
            T_M ← T.
            τ̂ ← c. // The exact counts at T_M are saved here.
            ExactCnt ← false.
        end
        ReplaceEdge-M(e).
    end

end
```

4 Experiments

We present experimental results to answer the following questions:

Q1 (**Accuracy**) How accurate and memory-efficient is FURL for local triangle counting in a multigraph stream? (Section 4.2)
Q2 (**Parameter**) How does the performance of FURL change with varying parameters? (Section 4.3)
Q3 (**Types of Anomaly**) Which types of anomalies does FURL find by utilizing both binary and weighted counting? (Section 4.4)
Q4 (**Discovery**) What are the discoveries from a real-world Bitcoin graph by FURL? (Section 4.5)
Q5 (**Scalability**) How scalable is FURL for local triangle counting in a multigraph stream? (Section 4.6)

4.1 Experimental Settings

Dataset. The real world graph datasets used in our experiments are listed in Table 3. We remove self-loops and edge direction from graphs and use a random order of edges for datasets.

Environment. The experiments are performed on a server with a single Intel Xeon E5-2630 v4 CPU (2.2GHz) and 256GB memory, and all methods used in this experiments are implemented in C++.

Competitors. We compare our proposed methods with three up-to-date methods: MultiMascot (Lim et al., 2018), TRIEST (Stefani et al., 2017), and PartitionCT (Wang et al., 2017). We refer to MultiMascot for binary and weighted counting as $\text{MultiMascot}_\text{B}$ and $\text{MultiMascot}_\text{W}$, respectively. We compare FURL_W and FURL-0_W with the two competing methods $\text{MultiMascot}_\text{W}$ and TRIEST

Table 3: Datasets used in our experiments.

Name	Node	Edge	Description
Livejournal[1]	4,847,571	68,475,391	Social network of LiveJournal
Bitcoin[2]	6,297,539	28,143,065	Bitcoin transaction network
DBLP[1]	1,314,050	18,986,618	Co-author network in DBLP
$\mathsf{Wiki}_{\mathsf{GER}}$[1]	506,174	4,555,759	Communication network of the German Wikipedia
Enron[1]	86,978	1,134,990	Enron email network
Facebook[1]	45,813	855,542	Wall posts on other user's wall on Facebook

[1] http://konect.uni-koblenz.de/networks/ [2] http://compbio.cs.uic.edu/data/bitcoin/

in weighted counting. We also compare FURL_B and FURL-0_B with the two competing methods $\text{MultiMascot}_\text{B}$ and PartitionCT in binary counting. TRIEST and PartitionCT are excluded for binary and weighted counting experiments respectively since they do not support each counting approach.

Evaluation Metric. To evaluate local triangle counting algorithms, we consider the following metrics.

- **Mean of Relative Error** (MRE): It measures how close an estimation τ_u is to the ground truth Δ_u in local triangle counting; $\mathsf{MRE} = \frac{1}{|V|}\sum_{u\in V}\frac{|\tau_u-\Delta_u|}{\Delta_u}$, where V is a set of nodes appearing in a graph stream whose number of triangles is larger than 0.
- **Running Time**: This measures how long an algorithm takes to process a multigraph in seconds.
- **Proportion ξ of Sampled Edges**: This is the dominant factor of required memory spaces; $\xi = \frac{M}{u}$, where M is the number of sampled edges and u is the number of unique edges in a graph.

For all the algorithms, all the metrics are computed by the average of The memory usage ξ is determined each time as follows. First, we run MultiMascot with a given edge sampling rate p and measure M of MultiMascot. Then, we set the memory sizes of PartitionCT, TRIEST, and FURL to the measured M.

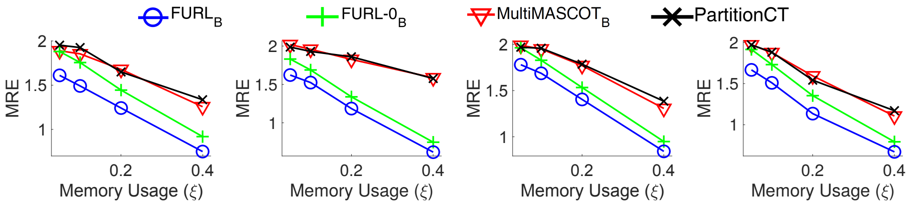
Parameters. We set the edge sampling rate $p \in \{0.05, 0.1, 0.2, 0.4\}$ in MultiMascot, the bucket size $J = m/10$ where m is the number of edges and the decaying factor $\delta = 0.7$ in FURL.

4.2 Accuracy and Memory-Efficiency of FURL

We compare accuracy and memory-efficiency of FURL and FURL-0 with those of competing methods MultiMascot, TRIEST, and PartitionCT showing the average of MRE in Enron, $\mathsf{Wiki}_{\mathsf{GER}}$, DBLP, and Livejournal (listed in Table 3).

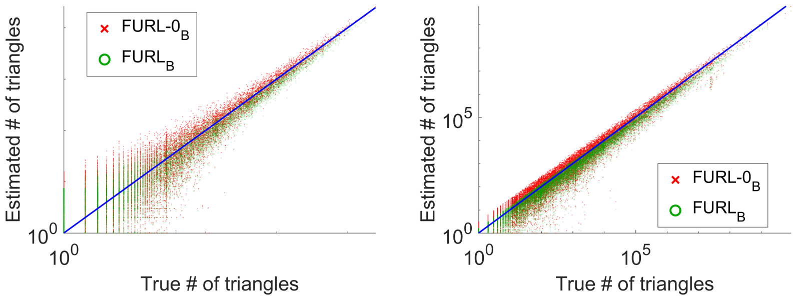
Fig. 1 shows the comparison between FURL_W, FURL-0_W, TRIEST, and $\text{MultiMascot}_\text{W}$ for weighted counting in MRE over the memory usage ξ. Note that our proposed method FURL_W outperforms the competing methods. The MRE of FURL_W is $1.23\times \sim 4.33\times$ and $1.15\times \sim 4.19\times$ smaller than that of $\text{MultiMascot}_\text{W}$ and TRIEST, respectively. Also, the MRE of FURL_W is $1.02\times \sim 1.45\times$ smaller than that of FURL-0_W. In terms of memory efficiency, FURL_W requires the smallest memory usage for a given error level.

Fig. 2 shows the comparison between FURL_B, FURL-0_B, PartitionCT, and $\text{MultiMascot}_\text{B}$ for binary counting in MRE over the memory usage ξ. Note that FURL_B outperforms $\text{MultiMascot}_\text{B}$ and PartitionCT. The MRE of FURL_B is $1.16\times \sim 2.57\times$ and $1.16\times \sim 2.55\times$ smaller than that of $\text{MultiMascot}_\text{B}$ and PartitionCT, respectively. The MRE of FURL_B is $1.08\times \sim 1.23\times$ smaller than that



(a) Enron (b) Wiki_{GER} (c) DBLP (d) Livejournal

Fig. 2: Mean of relative error (MRE) vs. memory usage of FURL_{B}, FURL-0_{B} and two competing methods $\text{MultiMascot}_{\text{B}}$ and PartitionCT for binary local triangle counting in a multigraph stream. Our proposed method FURL_{B} outperforms $\text{MultiMascot}_{\text{B}}$ and PartitionCT. The MRE of FURL_{B} is $1.16\times \sim 2.57\times$ and $1.16\times \sim 2.55\times$ smaller than that of $\text{MultiMascot}_{\text{B}}$ and PartitionCT, respectively, and is $1.08\times \sim 1.23\times$ smaller than that of FURL-0_{B}.



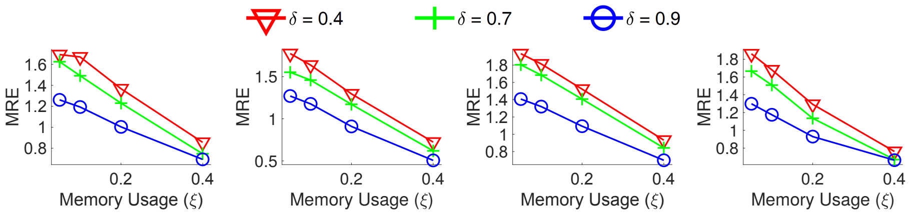
(a) binary counting (b) weighted counting

Fig. 3: Estimations of FURL (green points) and FURL-0 (red points) vs. the true number of triangles of each node in Enron. We set $\xi = 0.4$, $\delta = 0.7$ and $J = m/10$. The blue line indicates $y = x$ line. Most estimations of FURL are closer to the true number of triangles than those of FURL-0. Also, the variance of FURL is smaller than that of FURL-0.

of FURL-0_{B}. In terms of memory efficiency, as in the case of weighted counting, FURL_{B} requires the smallest memory usage for a given error level.

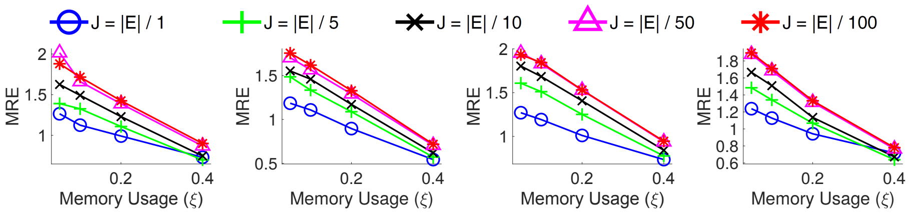
There are two main reasons why FURL outperforms the competitors. First, FURL uses memory efficiently and thus uses more triangles for estimation. Compared to Mascot, FURL fully uses the fixed memory space as soon as possible since FURL samples edges unconditionally until its buffer is full. However, Mascot uses the fixed memory space fully only when the entire graph stream is received. PartitionCT takes too long time to fully use the fixed memory space: based on coupon collector's theorem (Feller, 1968), it requires MH_M time to fully use the fixed memory space, where M is buffer size and H_M is the M-th harmonic number. Also, when existing sampled triangles are evicted from memory, FURL keeps the triangle counts for estimation, while PartitionCT excludes them. Thus, FURL uses more triangles for estimation than PartitionCT. TRIEST wastes memory space unnecessarily to deal with duplicate edges (see Section 2.1). Second, FURL improves the accuracy by reducing variance, as discussed in Section 3.2.

We compare the estimations of FURL and FURL-0 for each node to show how FURL gets better accuracy. Fig. 3 shows estimations of FURL and FURL-0 vs. the true number of triangles for each node in Enron. We set $\xi = 0.4$, $\delta = 0.7$



(a) Enron (b) Wiki_{GER} (c) DBLP (d) Livejournal

Fig. 4: Accuracy of FURL_B varying the decaying factor $\delta \in \{0.4, 0.7, 0.9\}$ with the bucket size $J = m/10$. The larger δ is, the higher the accuracy is in general.



(a) Enron (b) Wiki_{GER} (c) DBLP (d) Livejournal

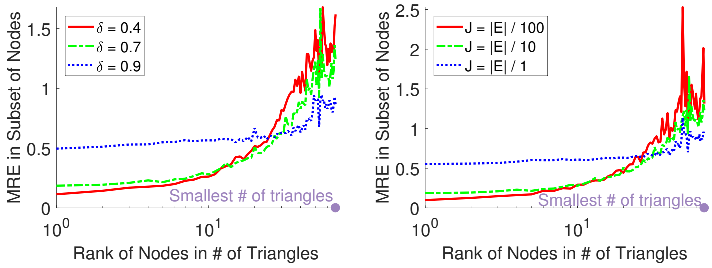
Fig. 5: Accuracy of FURL_B varying the bucket size $J \in \{|E|/1, |E|/5, |E|/10, |E|/50, |E|/100\}$ with the decaying factor $\delta = 0.7$. The larger J is,the higher the accuracy is in general.

and $J = m/10$. It shows how close estimations are to the true number of triangles. The blue line is $y = x$ where the true and estimated number of triangles are the same. Green and red points are estimations of FURL and FURL-0, respectively. Note that most estimations of FURL are closer to the true number of triangles than those of FURL-0 in both binary and weighted counting. In the quantitative aspect, MRE of FURL_B and FURL_W are $1.23\times$ and $1.2\times$ smaller than that of FURL-0_B and FURL-0_W respectively. Also, the variance of FURL is smaller than that of FURL-0 since estimations of FURL are more concentrated than those of FURL-0. Relative errors of estimations of FURL_B and FURL_W are $3.11\times$ and $1.93\times$ smaller than those of FURL-0_B and FURL-0_W respectively.

4.3 Performance of FURL Varying Parameters

We present experimental results varying parameters δ and J of FURL. We set $\xi = \{0.05, 0.1, 0.2, 0.4\}$. The decaying factor δ determines the weight of past estimations in weighted averaging. The bucket size J determines how often we do weighted averaging. Fig. 4 shows the accuracy of FURL_B varying the decaying factor $\delta \in \{0.4, 0.7, 0.9\}$ with the bucket size $J = m/10$. The larger δ is, the higher the accuracy is in general. Fig. 5 shows the accuracy of FURL_B varying the decaying factor $J \in \{m/i | i = 1, 5, 10, 50, 100\}$ with the decaying factor $\delta = 0.7$. In general, as J gets larger, the accuracy gets higher.

We observe that this improvement is from nodes having a small number of triangles. Note that the accuracy is improved for nodes with a small number of triangles, but degraded for those with a large number of triangles when δ and J increase. For most nodes with a small number of triangles, the accuracy depends on variance more significantly than unbiasedness. Thus, increasing δ and J results in the reduced error since larger δ and J imply smaller variance despite a slight bias. For nodes with a large number of triangles, however, larger δ and J can increase the error since unbiased estimation becomes more important than variance in such cases. This is because its large number of triangles has a similar effect to



(a) Varying δ (b) Varying J

Fig. 6: MRE over node ranks in the number of triangles by $\mathrm{FURL_B}$. A higher rank, corresponding to a smaller x-axis value, means a larger number of triangles. Note that large δ and J raise the error for few nodes having a large number of triangles, but for the remaining large portion of nodes, they reduce the error significantly. Hence, the overall error decreases for large δ and J as shown in Figures 4 and 5.

many samples for an estimation: i.e., the resulting variance is reduced. Thus, the increased bias by larger parameters makes the error increase. Yet, the overall error generally decreases since real-world graphs have many low degree nodes with a small number of triangles, but few high degree nodes with many triangles due to their skewed degree distributions.

Fig. 6 shows this difference in error over ranks of nodes in the number of triangles by $\mathrm{FURL_B}$ for $\xi = 0.4$ varying the parameters $\delta \in \{0.4, 0.7, 0.9\}$ and $J \in \{|E|/1, |E|/10, |E|/100\}$ in Enron. The default setting is $\delta = 0.7$ and $J = |E|/10$. We exclude nodes having no triangle, sort the remaining nodes in the decreasing order of their triangles, divide the range into 300 equal-sized intervals, and calculate MRE for nodes in each interval. Note that in the figure, as an x-axis value gets smaller, a rank gets higher which corresponds to a larger number of triangles. As shown in the figure, the error gets larger as δ and J becomes larger only for few nodes in very high ranks (rank < 10).

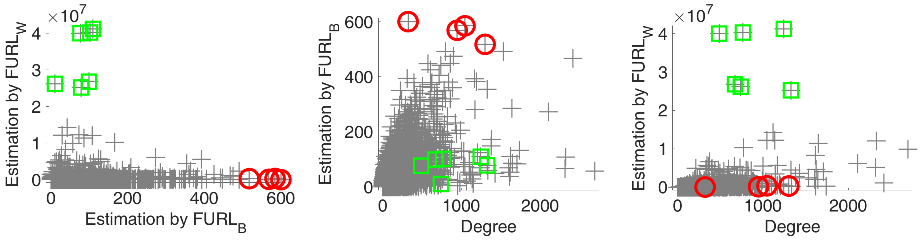
4.4 Anomaly Detection by Binary and Weighted Counting

We demonstrate two types of anomalous nodes discovered by $\mathrm{FURL_B}$ and $\mathrm{FURL_W}$ on Facebook (listed in Table 3). We set $\xi = 0.4$. There are anomalies which have too many triangles in binary counting but few triangles in weighted counting (marked red in Fig. 7) and vice versa (marked green in Fig. 7). To detect such anomalies, we compare the number of local triangles estimated by $\mathrm{FURL_B}$ and that by $\mathrm{FURL_W}$. However, the state-of-the-art methods PartitionCT and TRIEST cannot provide both and thus we compare the degree of nodes and the number of local triangles estimated by each method.

Fig .7a shows the comparison of the number of estimated triangles by $\mathrm{FURL_W}$ and $\mathrm{FURL_B}$. Note that FURL easily discovers the red nodes since they abnormally have a lot of triangles in binary counting compared to what they have in weighted counting. However, TRIEST cannot find the red nodes since it does not support binary counting, and the red nodes look normal in weighted triangle count vs. degree plot (Fig 7c). Likewise, FURL easily discovers the green nodes, but PartitionCT cannot (see Fig 7b).

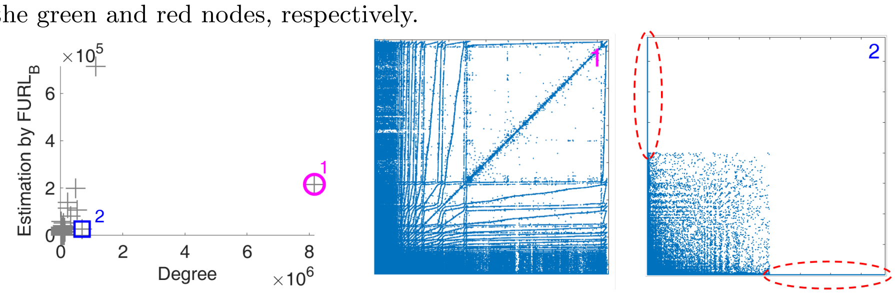
4.5 Anomaly Detection on Bitcoin Network

We investigate anomalous nodes discovered by FURL in a real-world Bitcoin network (listed in Table 3) where each node is an account and an edge (u, v) denotes at least one transaction between u and v. We use $\mathrm{FURL_B}$ and set $\xi = 0.4$.



(a) FURL (b) PARTITIONCT (c) TRIEST

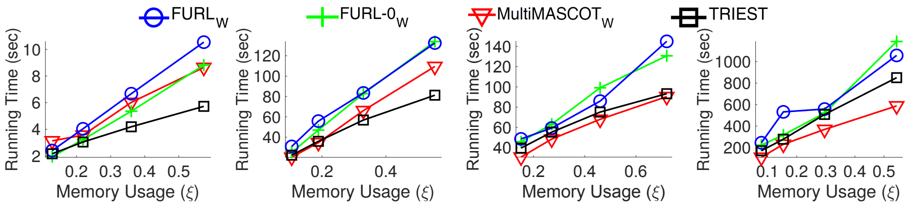
Fig. 7: Anomaly detection by FURL, PARTITIONCT, and TRIEST. There are anomalies which have too many triangles in binary counting but few triangles in weighted counting (marked red), and vice versa (marked green). FURL easily discovers the red and green nodes, but PARTITIONCT and TRIEST cannot find the green and red nodes, respectively.



(a) Degree vs. # of local triangles by FURL_B (b) Spy plot for egonet of the purple node (c) Spy plot for egonet of the blue node

Fig. 8: (a) Degree vs. the number of local triangles estimated by FURL_B in Bitcoin. Each point in the figure denotes an account in Bitcoin. The purple and blue points are marked as anomalous by FURL_B since they have few triangles compared to their degrees. It turns out the purple point belongs to a gambling site, and the blue point belongs to a Bitcoin mining pool; both of which behave differently compared to normal users. (b, c) Spy plots for egonetworks of the purple and blue nodes. The densities of nonzeros are small (1.61×10^{-5} and 1.02×10^{-4}, respectively), meaning that the neighbors are sparsely connected to each other. Note also that the spyplots show a near-star structure, which leads to few triangles in the egonets.

In Bitcoin, there are two anomalous accounts (marked as purple and blue points in Fig. 8a) with large degrees and few triangles: i.e., they make a lot of transactions to their neighbors, but the neighbors make few transactions among them. We investigate the two accounts at https://blockchain.info/tags that informs each account's corresponding website if there is any. It turns out the purple point belongs to a gambling site, and the blue point belongs to a Bitcoin mining pool. In both egonets, neighbors are sparsely connected to each other since any two accounts in them are not likely to know each other by nature. In the purple point, the account for the gambling site makes transactions to diverse people that do not interact with each other frequently. In the blue point, the account for the Bitcoin mining pool also makes transactions to people that do not make transactions with each other. Note that a Bitcoin mining pool is used by Bitcoin miners to pool their computing power and distribute the reward according to the amount they contributed; diverse identities of Bitcoin miners lead to diverse neighborhoods.



(a) Enron (b) Wiki$_{\mathsf{GER}}$ (c) DBLP (d) Livejournal

Fig. 9: Running time vs. memory usage of FURL$_{\mathrm{W}}$, FURL-0$_{\mathrm{W}}$, and two competing methods MultiMascot$_{\mathrm{W}}$ and TRIEST for weighted local triangle counting. In general, FURL is slower than competitors due to its use of priority queue data structure: e.g., FURL$_{\mathrm{W}}$ is $1.11\times \sim 2.31\times$ slower than MultiMascot$_{\mathrm{W}}$ except at $\xi = 0.05$ in Enron and $1.07\times \sim 1.92\times$ slower than TRIEST. However, FURL still shows linear scalability and gives the best accuracy as shown in Figure 1.

Fig. 8b, 8c show spy plots for egonetworks of the purple and blue nodes. The densities of nonzeros are 1.61×10^{-5} and 1.02×10^{-4}, respectively, meaning that the neighbors are sparsely connected to each other. Note that the spy plots show a near-star structure, where most nodes are connected only to few central nodes. Especially, the latter half of nodes (marked in a dotted red ellipse) in Fig. 8c are connected only to the central node of the egonet, leading to small triangles.

4.6 Running Time of FURL

We measure the running time of FURL, FURL-0, TRIEST, and MultiMascot in Enron, Wiki$_{\mathsf{GER}}$, and DBLP.

Fig. 9 shows the running time of FURL$_{\mathrm{W}}$ and FURL-0$_{\mathrm{W}}$ compared to that of the competing methods TRIEST and MultiMascot$_{\mathrm{W}}$ in weighted counting. The running time of FURL$_{\mathrm{W}}$ is close to that of FURL-0$_{\mathrm{W}}$ since they are almost the same except for the weighted averaging process. FURL$_{\mathrm{W}}$ is $1.11\times \sim 2.31\times$ slower than MultiMascot$_{\mathrm{W}}$ except at $\xi = 0.05$ in Enron, and $1.07\times \sim 1.92\times$ slower than TRIEST; the reason is that MultiMascot$_{\mathrm{W}}$ does not use a buffer and TRIEST uses an array as the buffer, while FURL$_{\mathrm{W}}$ and FURL-0$_{\mathrm{W}}$ use a priority queue as the buffer to keep an edge with the Mth smallest hash value. The result for binary counting shows a similar trend as that of Fig. 9. However, FURL still shows linear scalability, and has a superior accuracy compared to the existing methods, as described in Section 4.2.

5 Conclusion

We propose FURL, a memory-efficient and accurate algorithm for local triangle estimation in a multigraph stream. FURL processes a multigraph stream with a fixed number of sampled edges. FURL provides superior accuracy by reducing the variance of estimation via regularization, and sampling more triangles. Also, FURL finds anomalies which the state-of-the-art methods cannot find. Experimental results demonstrate that FURL provides the best accuracy compared to the state-of-the-art algorithm in a memory-efficient way. Using FURL on a Bitcoin transaction network, we discover interesting accounts which are used by gambling and Bitcoin mining pool sites. Future works include extending FURL to fully dynamic graphs streams which include both edge insertions and deletions.

Acknowledgements This work was supported by AFOSR/AOARD under the Grant No. FA2386-16-1-4044. This work was also supported by Institute for Institute of Information & Communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) (No.R0190-15-2012, High Performance Big Data Analytics Platform Performance

Acceleration Technologies Development). The Institute of Engineering Research at Seoul National University provided research facilities for this work. The ICT at Seoul National University provides research facilities for this study.

References

Alon N, Yuster R, Zwick U (1997) Finding and counting given length cycles. Algorithmica 17(3):209–223, https://doi.org/10.1007/BF02523189

Becchetti L, Boldi P, Castillo C, Gionis A (2010) Efficient algorithms for large-scale local triangle counting. ACM Transactions on Knowledge Discovery from Data 4(3):13:1–13:28, https://doi.org/10.1145/1839490.1839494

Berry JW, Hendrickson B, LaViolette RA, Phillips CA (2011) Tolerating the community detection resolution limit with edge weighting. Physical Review E 83(5):056119, https://doi.org/10.1103/PhysRevE.83.056119

Broder AZ, Charikar M, Frieze AM, Mitzenmacher M (2000) Min-wise independent permutations. Journal of Computer and System Sciences 60(3):630–659, https://doi.org/10.1006/jcss.1999.1690

Chou B, Suzuki E (2010) Discovering community-oriented roles of nodes in a social network. In: 12th International Conference Data Warehousing and Knowledge Discovery (DAWAK), pp 52–64

Eckmann JP, Moses E (2002) Curvature of co-links uncovers hidden thematic layers in the world wide web. Proceedings of the national academy of sciences 99(9):5825–5829, https://doi.org/10.1073/pnas.032093399

Epasto A, Lattanzi S, Mirrokni VS, Sebe I, Taei A, Verma S (2015) Ego-net community mining applied to friend suggestion. Proceedings of the VLDB Endowment 9(4):324–335, http://doi.org/10.14778/2856318.2856327

Feller W (1968) An Introduction to Probability Theory and Its Applications, vol 1. Wiley

Flajolet P, Martin GN (1985) Probabilistic counting algorithms for data base applications. Journal of Computer and System Sciences 31(2):182–209, https://doi.org/10.1016/0022-0000(85)90041-8

Gentle JE (2009) Computational Statistics, 1st edn. Springer Publishing Company, Incorporated

Jha M, Pinar A, Seshadhri C (2015) Counting triangles in real-world graph streams: Dealing with repeated edges and time windows. In: 49th Asilomar Conference on Signals, Systems, and Computers (ACSSC), pp 1507–1514

Kutzkov K, Pagh R (2013) On the streaming complexity of computing local clustering coefficients. In: Sixth ACM International Conference on Web Search and Data Mining, (WSDM), pp 677–686

Latapy M (2008) Main-memory triangle computations for very large (sparse (power-law)) graphs. Theoretical Computer Science 407(1-3):458–473, https://doi.org/10.1016/j.tcs.2008.07.017

Lim Y, Kang U (2015) 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 (KDD), pp 685–694

Lim Y, Jung M, Kang U (2018) Memory-efficient and accurate sampling for counting local triangles in graph streams: From simple to multigraphs. ACM Transactions on Knowledge Discovery from Data 12(1):4:1–4:28, https://doi.org/10.1145/3022186

Milo R, Shen-Orr S, Itzkovitz S, Kashtan N, Chklovskii D, Alon U (2002) Network motifs: simple building blocks of complex networks. Science 298(5594):824–827

Pagh R, Tsourakakis CE (2012) Colorful triangle counting and a mapreduce implementation. Information Processing Letters 112(7):277–281, https://doi.org/10.1016/j.ipl.2011.12.007

Stefani LD, Epasto A, Riondato M, Upfal E (2016) Trièst: Counting local and global triangles in fully-dynamic streams with fixed memory size. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), pp 825–834

Stefani LD, Epasto A, Riondato M, Upfal E (2017) Trièst: Counting local and global triangles in fully dynamic streams with fixed memory size. ACM Transactions on Knowledge Discovery from Data 11(4):43:1–43:50, https://doi.org/10.1145/3059194

Sunter A (1977) List sequential sampling with equal or unequal probabilities without replacement. Applied Statistics 26(3):261–268, http://doi.org/10.2307/2346966

Suri S, Vassilvitskii S (2011) Counting triangles and the curse of the last reducer. In: Proceedings of the 20th international conference on World wide web (WWW), pp 607–614

Tsourakakis CE, Kang U, Miller GL, Faloutsos C (2009) DOULION: counting triangles in massive graphs with a coin. In: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD) 1, 2009, pp 837–846

Vitter JS (1985) Random sampling with a reservoir. ACM Trans Math Softw 11(1):37–57, https://doi.org/10.1145/3147.3165

Wang P, Qi Y, Sun Y, Zhang X, Tao J, Guan X (2017) Approximately counting triangles in large graph streams including edge duplicates with a fixed memory usage. Proceedings of the VLDB Endowment 11(2):162–175, http://doi.org/10.14778/3149193.3149197

Welser HT, Gleave E, Fisher D, Smith MA (2007) Visualizing the signatures of social roles in online discussion groups. Journal of Social Structure 8(2):1–32

Yang Z, Wilson C, Wang X, Gao T, Zhao BY, Dai Y (2011) Uncovering social network sybils in the wild. In: Proceedings of the 11th ACM SIGCOMM Internet Measurement Conference, (IMC), pp 259–268

Appendix A Proofs of Lemmas and Theorems

Theorem 1 *Let Δ_u be the true local triangle count for a node u, and c_u be the estimation given by* FURL-0$_\mathrm{B}$. *For every node u, $\mathbb{E}[c_u] = \Delta_u$.*

Proof Let T_λ be the time λ is formed and T_M be the last time we have the exact triangle counts. Let Λ_u^- be the set of triangles containing node u that are formed when we have the exact triangle counts, i.e. $T_\lambda \leq T_M$, and Λ_u^+ be the set of triangles containing node u that are formed when we do not have the exact triangle counts, i.e. $T_\lambda > T_M$. Note that every triangle containing node u is included in either Λ_u^- or Λ_u^+. We define random variables X_λ^- and X_λ^+ as follows:

$$X_\lambda^- = \begin{cases} 1 & \lambda \textit{ is counted} \\ 0 & \textit{otherwise.} \end{cases} \quad X_\lambda^+ = \begin{cases} q_{T_\lambda} = \frac{M-3}{M} \cdot \frac{1}{h_{max}^3} & \lambda \textit{ is counted} \\ 0 & \textit{otherwise.} \end{cases}$$

For a triangle $\lambda \in \Lambda_u^-$, $\mathbb{E}\left[X_\lambda^-\right] = 1 \times \Pr\left[\lambda \textit{ is counted}\right] = 1$ since all edges are unconditionally sampled at $T \leq T_M$.

Now we show $\mathbb{E}\left[X_\lambda^+\right] = 1$ for each triangle $\lambda \in \Lambda_u^+$. Let $u(T) \geq M$ be the number of unique edges that have arrived so far in the stream at time T. Considering repeated edges as one edge, we get a refined stream that can be viewed as a sequence of independent random variables $h_i (1 \leq i \leq u(T_\lambda))$ that has a uniform distribution in range $(0, 1)$. h_{max} can be viewed as the Mth smallest value among $u(T_\lambda)$ number of random variables since D keeps the M minimum values. Then $h_{max} \sim Beta(M, u(T_\lambda) + 1 - M)$ by the rule of kth order statistic in uniform distribution (Gentle, 2009). Thus its probability density function $f(x) = \frac{\gamma(u(T_\lambda)+1)}{\gamma(M)\cdot\gamma(u(T_\lambda)+1-M)} \cdot x^{M-1} \cdot (1-x)^{u(T_\lambda)-M}$. $\Pr\left[\lambda \textit{ is counted}\right] = \frac{M(M-1)(M-2)}{u(T_\lambda)(u(T_\lambda)-1)(u(T_\lambda)-2)}$ since it is the probability that 3 edges of λ are stored in the buffer D simultaneously. Note that $\Pr\left[\lambda \textit{ is counted}\right]$ is independent of the event $h_{max} = x$.

$$\begin{aligned} \mathbb{E}\left[X_\lambda^+\right] &= \mathbb{E}\left[\mathbb{E}\left[X_\lambda^+ | h_{max}\right]\right] \\ &= \int_0^1 \Pr\left[\lambda \textit{ is counted} | h_{max} = x\right] \cdot q_{T_\lambda} \cdot f(x) dx \\ &= \frac{M(M-1)(M-2)}{u(T_\lambda)(u(T_\lambda)-1)(u(T_\lambda)-2)} \cdot \int_0^1 \frac{M-3}{M} \cdot \frac{1}{x^3} \cdot f(x) dx = 1 \end{aligned}$$

Therefore, $\mathbb{E}[c_u] = \sum_{\lambda \in \Lambda_u^-} \mathbb{E}\left[X_\lambda^-\right] + \sum_{\lambda \in \Lambda_u^+} \mathbb{E}\left[X_\lambda^+\right] = \Delta_u$.

■

Theorem 2 *Let Δ_u be the true local triangle count for a node u, and c_u be the estimation given by* FURL-0$_\mathrm{W}$. *For every node u, $\mathbb{E}[c_u] = \Delta_u$.*

Proof We define random variables X_λ^- and X_λ^+ which have 1 and $q_{T_\lambda} = \frac{M-2}{M} \cdot \frac{1}{h_{max}^2}$ respectively if λ is counted, or 0 otherwise.

$$X_\lambda^- = \begin{cases} 1 & \lambda \textit{ is counted} \\ 0 & \textit{otherwise.} \end{cases} \quad X_\lambda^+ = \begin{cases} q_{T_\lambda} = \frac{M-2}{M} \cdot \frac{1}{h_{max}^2} & \lambda \textit{ is counted} \\ 0 & \textit{otherwise.} \end{cases}$$

Then, $\mathbb{E}\left[c_u\right] = \sum_{\lambda \in \Lambda_u^-} \mathbb{E}\left[X_\lambda^-\right] + \sum_{\lambda \in \Lambda_u^+} \mathbb{E}\left[X_\lambda^+\right]$.

In FURL-0$_\mathrm{W}$, although λ updates the estimation, the buffer does not meet a new edge at T_λ yet since a triangle is updated before sampling the edge. Then h_{max} can be viewed as the Mth smallest value among $u(T_\lambda - 1)$ number of random variables in range $(0, 1)$ and $h_{max} \sim Beta(M, u(T_\lambda - 1) + 1 - M)$. Thus its probability density function is given as $g(x) = \frac{\gamma(u(T_\lambda-1)+1)}{\gamma(M)\cdot\gamma(u(T_\lambda-1)+1-M)} \cdot x^{M-1} \cdot (1-x)^{u(T_\lambda-1)-M}$.

The probability that a triangle λ is counted is $\frac{M(M-1)}{u(T_\lambda-1)(u(T_\lambda-1)-1)}$ because λ is counted if an arriving edge e forms λ with the other two edges in the buffer regardless of sampling e.

$$
\begin{aligned}
\mathbb{E}\left[X_\lambda^+\right] &= \Pr\left[\lambda \text{ } is\ counted\right] \cdot \int_0^1 q_{(T_\lambda-1)} \cdot g(x)dx \\
&= \frac{M(M-1)}{u(T_\lambda-1)(u(T_\lambda-1)-1)} \cdot \int_0^1 \frac{M-2}{M} \cdot \frac{1}{x^2} \cdot g(x)dx \\
&= \int_0^1 \frac{\gamma(u(T_\lambda-1)-1)}{\gamma(M-2)\gamma(u(T_\lambda-1)+1-M)} \cdot x^{M-3}(1-x)^{u(T_\lambda-1)-M}dx = 1
\end{aligned}
$$

Therefore, $\mathbb{E}\left[c_u\right] = \sum_{\lambda \in \Lambda_u^-} \mathbb{E}\left[X_\lambda^-\right] + \sum_{\lambda \in \Lambda_u^+} \mathbb{E}\left[X_\lambda^+\right] = \Delta_u$.

■

Lemma 1 *Let $b_\lambda > 0$ be the bucket where a triangle λ is formed and Y_λ be the estimated count of a triangle λ by* FURL$_\mathrm{B}$. *Then,*

$$\mathbb{E}\left[Y_\lambda\right] = 1 - \phi(b_\lambda),$$

where $\phi(i) = \delta^{b-i+1}$ and b is the current bucket.

Proof Let $W(i) = (1-\delta)\delta^{b-i}$ for $i \geq 1$ and $q_{T_\lambda} = \frac{(T-1)(T-2)}{M(M-1)}$. By definition, for λ appearing in bucket b_λ, Y_λ becomes $q_{T_\lambda}W(b_\lambda) + q_{T_\lambda}W(b_\lambda+1) + \cdots + q_{T_\lambda}W(b)$ if λ is counted; if λ is not counted, $Y_\lambda = 0$. Thus, we obtain

$$\mathbb{E}\left[Y_\lambda\right] = \Pr\left[\lambda \text{ } is\ counted\right] \cdot \left(q_{T_\lambda} \sum_{b_\lambda \leq j \leq b} W(j)\right) = 1 - \phi(b_\lambda).$$

■

Lemma 2 *Let $b_\lambda > 0$ be the bucket where a triangle λ is formed and Y_λ be the estimated count of a triangle λ by* FURL$_\mathrm{B}$. *Then,*

$$\mathrm{Var}\left[Y_\lambda\right] = \left(1 - \phi(b_\lambda)\right)^2 \left(k_{T_\lambda} - 1\right),$$

where T_λ is the first time all three edges of λ arrive, $k_{T_\lambda} = \frac{(M-3)(u(T_\lambda)-3)(u(T_\lambda)-4)(u(T_\lambda)-5)}{M(M-4)(M-5)(M-6)}$, $\phi(i) = \delta^{b-i+1}$, and b is the current bucket.

Proof Following the definition $\mathrm{Var}\left[Y_\lambda\right] = \mathbb{E}\left[Y_\lambda^2\right] - \mathbb{E}\left[Y_\lambda\right]^2$ with Lemma 1, the proof is done.

■

Theorem 3 *Let Y_λ and X_λ be the estimated counts of a triangle λ by* FURL_B *and* FURL-0_B*, respectively. Consider any triangle λ that is counted at time $T_\lambda > T_M$. Let $u(T)$ be the number of unique edges that have arrived at time T. If $u(T_\lambda) \geq \sqrt[3]{\frac{1+\alpha}{\alpha}}M + 3$, the interval by $\mathbb{E}[Y_\lambda] \pm \alpha \cdot \text{Var}[Y_\lambda]$ is strictly included in that by $\mathbb{E}[X_\lambda] \pm \alpha \cdot \text{Var}[X_\lambda]$ for any α.*

Proof We first show that $\mathbb{E}[Y_\lambda] - \alpha \cdot \text{Var}[Y_\lambda] > \mathbb{E}[X_\lambda] - \alpha \cdot \text{Var}[X_\lambda]$.

Let $\psi_\lambda = 1 - \delta^{b-b_\lambda+1}$; we will find the condition satisfying $\psi_\lambda - \alpha\psi_\lambda^2(k_{T_\lambda} - 1) - 1 + \alpha(k_{T_\lambda} - 1) > 0$, which is developed as follows.

$$\psi_\lambda > \frac{1 - \alpha k_{T_\lambda} + \alpha}{\alpha k_{T_\lambda} - \alpha}. \tag{1}$$

Below, we will show the condition under which Eq. (1) holds. Let $u(T_\lambda) - 3 = \beta M$. By definition,

$$k_{T_\lambda} > \frac{(M-3)(u(T_\lambda)-3)^3}{M(M-4)^3} > \frac{(u(T_\lambda)-3)^3}{M^3} = \beta^3.$$

Then,

$$\frac{1 - \alpha k_{T_\lambda} + \alpha}{\alpha k_{T_\lambda} - \alpha} < \frac{1 - \alpha\beta^3 + \alpha}{\alpha\beta^3 - \alpha} = \frac{1}{\alpha\beta^3 - \alpha} - 1. \tag{2}$$

Now we examine the left term of Eq. (1). Since $1 \leq b_\lambda \leq b$, the lower bound of ψ_λ becomes $\psi_\lambda \geq 1 - \delta$. Then, we obtain a sufficient condition for (1) as follows:

$$\beta^3 \geq \frac{1 + 2\alpha - \alpha\delta}{2\alpha - \alpha\delta} \tag{3}$$

For $\beta \geq \sqrt[3]{\frac{1+\alpha}{\alpha}}$, Eq. (3) always holds since the upper bound of the right term becomes $\frac{1+\alpha}{\alpha}$. Note that $0 \leq \delta < 1$. Thus, we finally obtain the condition under which Eq. (1) holds as follows:

$$u(T_\lambda) \geq \sqrt[3]{\frac{1+\alpha}{\alpha}}M + 3.$$

Due to the underestimation of FURL_B, $\mathbb{E}[Y_\lambda] + \alpha \cdot \text{Var}[Y_\lambda] < \mathbb{E}[X_\lambda] + \alpha \cdot \text{Var}[X_\lambda]$ always holds, which completes the proof.

■

Lemma 3 *Let b_λ be the bucket where λ is formed and Y_λ be the estimated count of a triangle λ with $b_\lambda > 0$ by* FURL_W*. For every triangle λ,*

$$\mathbb{E}[Y_\lambda] = 1 - \phi(b_\lambda),$$

where $\phi(i) = \delta^{b-i+1}$ and b is the current bucket.

Proof The lemma is proved in the same way as in Lemma 1.

■

Lemma 4 *Let b_λ be the bucket where λ is formed and Y_λ be the estimated count of a triangle λ with $b_\lambda > 0$ by* $\mathrm{FURL_W}$. *For every triangle λ,*

$$\mathrm{Var}\left[Y_\lambda\right] = \left(1-\phi(b_\lambda)\right)^2 \left(l_{T_\lambda} - 1\right),$$

where T_λ is the first time all three edges of λ arrive, $l_{T_\lambda} = \frac{(M-2)(u(T_\lambda-1)-2)(u(T_\lambda-1)-3)}{M(M-3)(M-4)}$, $\phi(i) = \delta^{b-i+1}$, *and b is the current bucket.*

Proof The lemma is proved in the same way as in Lemma 2.

■

Theorem 4 *Let Y_λ and X_λ be the estimated counts of a triangle λ by* $\mathrm{FURL_W}$ *and* FURL-0_{W}, *respectively. Consider any triangle λ that is counted at time $T_\lambda > T_M$. If $u(T_\lambda - 1) \geq \sqrt{\frac{1+\alpha}{\alpha}}M + 2$, the interval by $\mathbb{E}\left[Y_\lambda\right] \pm \alpha \cdot \mathrm{Var}\left[Y_\lambda\right]$ is strictly included in that by $\mathbb{E}\left[X_\lambda\right] \pm \alpha \cdot \mathrm{Var}\left[X_\lambda\right]$ for any α.*

Proof We first show that $\mathbb{E}\left[Y_\lambda\right] - \alpha \cdot \mathrm{Var}\left[Y_\lambda\right] > \mathbb{E}\left[X_\lambda\right] - \alpha \cdot \mathrm{Var}\left[X_\lambda\right]$.

Let $\psi_\lambda = 1 - \delta^{B-b_\lambda+1}$; we will find the condition satisfying:

$$\psi_\lambda > \frac{1 - \alpha l_{T_\lambda} + \alpha}{\alpha l_{T_\lambda} - \alpha}. \tag{4}$$

Let $u(T_\lambda - 1) - 2 = \beta M$. Then,

$$\frac{1 - \alpha l_{T_\lambda} + \alpha}{\alpha l_{T_\lambda} - \alpha} < \frac{1}{\alpha\beta^2 - \alpha} - 1. \ (\because\ l_{T_\lambda} > \frac{(u(T_\lambda-1)-2)^2}{M^2} = \beta^2.)$$

Since $1 \leq b_\lambda \leq B$, the lower bound of ψ_λ becomes $\psi_\lambda \geq 1 - \delta$. Then, we obtain a sufficient condition for (4) as follows:

$$\beta^2 \geq \frac{1 + 2\alpha - \alpha\delta}{2\alpha - \alpha\delta} \tag{5}$$

For $\beta \geq \sqrt{\frac{1+\alpha}{\alpha}}$, Eq. (5) always holds since the upper bound of the right term becomes $\frac{1+\alpha}{\alpha}$. Note that $0 \leq \delta < 1$. Thus, we finally obtain the condition under which Eq. (4) holds as follows:

$$u(T_\lambda - 1) \geq \sqrt{\frac{1+\alpha}{\alpha}}M + 2.$$

Due to the underestimation of $\mathrm{FURL_W}$, $\mathbb{E}\left[Y_\lambda\right] + \alpha \cdot \mathrm{Var}\left[Y_\lambda\right] < \mathbb{E}\left[X_\lambda\right] + \alpha \cdot \mathrm{Var}\left[X_\lambda\right]$ always holds, which completes the proof.

■

Lemma 5 *Let Δ_u be the true local triangle count for a node u and Y_u be the estimation given by* FURL *for node u. Then,*

$$(1-\delta)\,\Delta_u < \mathbb{E}\left[Y_u\right] \leq \Delta_u.$$

Proof Let X_λ and Y_λ be the estimated count of a triangle λ by FURL-0 and FURL respectively, T_λ be the time λ is formed, and T_M be the last time we have the exact triangle counts. Let Λ_u^- be the set of triangles containing node u that are formed when we have the exact triangle counts, i.e. $T_\lambda \leq T_M$, and Λ_u^+ be the set of triangles containing node u that are formed when we do not have the exact triangle counts, i.e. $T_\lambda > T_M$. Note that every triangle containing node u is included in either Λ_u^- or Λ_u^+.

For triangle $\lambda^- \in \Lambda_u^-$,

$$Y_{\lambda^-} = X_{\lambda^-}$$

and for triangle $\lambda^+ \in \Lambda_u^+$,

$$Y_{\lambda^+} = W(b_{\lambda^+})X_{\lambda^+} + W(b_{\lambda^+}+1)X_{\lambda^+} + \cdots + W(b)X_{\lambda^+} = \left(1 - \delta^{b-b_{\lambda^+}+1}\right) X_{\lambda^+}$$

where b_λ is the bucket where a triangle λ is formed and $W(i) = (1-\delta)\delta^{b-i}$. Then,

$$\begin{aligned}
\mathbb{E}\left[Y_u\right] &= \sum_{\lambda^-\in\Lambda_u^-} \mathbb{E}\left[X_{\lambda^-}\right] + \sum_{\lambda^+\in\Lambda_u^+} \mathbb{E}\left[\left(1-\delta^{b-b_{\lambda^+}+1}\right) X_{\lambda^+}\right] \\
&= \sum_{\lambda^-\in\Lambda_u^-} 1 + \sum_{\lambda^+\in\Lambda_u^+} \left(1-\delta^{b-b_{\lambda^+}+1}\right).
\end{aligned}$$

$1 \leq b_{\lambda^+} \leq b$ and $0 < \delta < 1$. Thus,

$$(1-\delta)\, \Delta_u < \mathbb{E}\left[Y_u\right] \leq \Delta_u.$$

■

Lemma 6 *Let X_u and Y_u be the estimated local triangle count for a node u given by* FURL-0 *and* FURL *respectively, Λ_u be the set of triangles containing node u and b_λ be the bucket where a triangle λ is formed.*

$$(1-\delta)^2 \operatorname{Var}\left[X_u\right] \leq \operatorname{Var}\left[Y_u\right] \leq \left(1-\delta^b\right)^2 \operatorname{Var}\left[X_u\right]$$

where b is the current bucket.

Proof Let X_λ and Y_λ be the estimated count of a triangle λ by FURL-0 and FURL respectively. Let Λ_u^- be the set of triangles containing node u that are formed when we have the exact triangle counts and Λ_u^+ be the set of triangles containing node u that are formed when we do not have the exact triangle counts.

For triangle $\lambda^- \in \Lambda_u^-$,

$$Y_{\lambda^-} - \mathbb{E}\left[Y_{\lambda^-}\right] = X_{\lambda^-} - \mathbb{E}\left[X_{\lambda^-}\right] = 1 - 1 = 0$$

and for triangle $\lambda^+ \in \Lambda_u^+$,

$$Y_{\lambda^+} - \mathbb{E}\left[Y_{\lambda^+}\right] = \left(1-\delta^{b-b_{\lambda^+}+1}\right) X_{\lambda^+} - \left(1-\delta^{b-b_{\lambda^+}+1}\right) \mathbb{E}\left[X_{\lambda^+}\right] = \left(1-\delta^{b-b_{\lambda^+}+1}\right) \left(X_{\lambda^+} - 1\right)$$

where b_λ is the bucket where a triangle λ is formed. Let Λ_u be the set of triangles containing node u. Then,

$$\begin{aligned}
\mathrm{Var}\left[Y_u\right] &= \sum_{\lambda_1 \in \Lambda_u} \sum_{\substack{\lambda_2 \in \Lambda_u, \\ \lambda_1 \neq \lambda_2}} \mathrm{Cov}\left[Y_{\lambda_1}, Y_{\lambda_2}\right] \\
&= \sum_{\lambda_1 \in \Lambda_u^+} \sum_{\substack{\lambda_2 \in \Lambda_u^+, \\ \lambda_1 \neq \lambda_2}} \mathrm{Cov}\left[Y_{\lambda_1}, Y_{\lambda_2}\right] \;\; (\because Y_{\lambda^-} - \mathbb{E}\left[Y_{\lambda^-}\right] = 0) \\
&= \sum_{\lambda_1 \in \Lambda_u^+} \sum_{\substack{\lambda_2 \in \Lambda_u^+, \\ \lambda_1 \neq \lambda_2}} \left(1 - \delta^{b - b_{\lambda_1} + 1}\right)\left(1 - \delta^{b - b_{\lambda_2} + 1}\right) \mathrm{Cov}\left[X_{\lambda_1}, X_{\lambda_2}\right].
\end{aligned}$$

$1 \leq b_{\lambda^+} \leq b$ and $0 < \delta < 1$. Thus,

$$(1-\delta)^2 \,\mathrm{Var}\left[X_u\right] \leq \mathrm{Var}\left[Y_u\right] \leq \left(1 - \delta^b\right)^2 \mathrm{Var}\left[X_u\right].$$

$\blacksquare$

Theorem 5 *Let $u(T)$ be the number of unique edges that have arrived at time T. If $\delta^b > 1 - \sqrt{1 - \frac{\mathbb{E}[X_u]}{\alpha \mathrm{Var}[X_u]}}$, the interval by $\mathbb{E}\left[Y_\lambda\right] \pm \alpha \cdot \mathrm{Var}\left[Y_\lambda\right]$ is strictly included in that by $\mathbb{E}\left[X_\lambda\right] \pm \alpha \cdot \mathrm{Var}\left[X_\lambda\right]$ for any α.*

Proof We first show that $\mathbb{E}\left[Y_\lambda\right] - \alpha \cdot \mathrm{Var}\left[Y_\lambda\right] > \mathbb{E}\left[X_\lambda\right] - \alpha \cdot \mathrm{Var}\left[X_\lambda\right]$. By Lemmas 5 and 6,

$$(1-\delta)\,\Delta_u - \left(1 - \delta^b\right)^2 \alpha \mathrm{Var}\left[X_u\right] > \Delta_u - \alpha \mathrm{Var}\left[X_u\right]$$

which is developed as follows.

$$0 > \delta^{2b} - 2\delta^b + \frac{\delta \Delta_u}{\alpha \mathrm{Var}\left[X_u\right]}.$$

$0 < \delta < 1$. Then, we obtain a sufficient condition as follows:

$$1 - \frac{\Delta_u}{\alpha \mathrm{Var}\left[X_u\right]} > \left(\delta^b - 1\right)^2.$$

Thus, we finally obtain the condition as follows:

$$\delta^b > 1 - \sqrt{1 - \frac{\Delta_u}{\alpha \mathrm{Var}\left[X_u\right]}}.$$

Due to the underestimation of FURL-0, $\mathbb{E}\left[Y_\lambda\right] + \alpha \cdot \mathrm{Var}\left[Y_\lambda\right] < \mathbb{E}\left[X_\lambda\right] + \alpha \cdot \mathrm{Var}\left[X_\lambda\right]$ always holds, which completes the proof.

$\blacksquare$