Triangle Counting by Adaptively Resampling over Evolving Graph Streams

Wei Xuan†‡ and Huawei Cao† and Mingyu Yan† and Zhimin Tang† and Xiaochun Ye† and Dongrui Fan†
†Institute of Computing Technology, Chinese Academy of Sciences, China
‡University of Chinese Academy of Sciences, China
Email: {xuanwei, caohuawei, yanmingyu, tang, yexiaochun, fandr}@ict.ac.cn

***Abstract*—Triangle counting is a fundamental graph mining problem, widely used in many real-world application scenarios. Due to the large scale of graph streams and limited memory space, it is appropriate to achieve the estimation of global and local triangles by sampling. Existing streaming algorithms for triangle counting can be generalized into two categories. One is Reservoir-based methods employing a fixed memory budget, whose size is difficult to set for accurate estimation without any prior knowledge about graph streams. The other is Bernoulli-based methods, which sample edges by a given probability with uncontrollable memory budget. In this work, we propose a novel and bounded-sampling-ratio method, called BSR-Sample, by adaptively resizing memory budget upwards over evolving graph streams. BSR-Sample can keep the sampling ratio always greater than or equal to a specified threshold with available memory space. Then, we design BSR-TC, a single-pass streaming algorithm for both global and local triangle counting, based on BSR-Sample. Experimental results show that BSR-TC achieves accuracy of at least 99.8% for global triangles, when the ratio of initial memory budget to whole graph streams $\geq$ 0.002% and given threshold = 20%. And our proposed BSR-TC can gain more advantage than the state-of-the-art algorithms over the continuous growth of graph streams.**

***Index Terms*—Evolving Graph Streams, Triangle Counting, Bounded Sampling Ratio**

I. Introduction

There has been a burst interest in graph streams in recent decades, covering lots of real-world application scenarios, such as social networks, E-commerce, traffic networks [1], [2]. Graph streams are a continuous sequence of data items, often abstracted as edge streams, expressing entities and the relationships between these entities [3]. Triangle counting has many important applications and is widely used in graph data mining. For instance, the number of triangles can be used to detect communities [4], study clustering coefficient and connectivity [5] of social networks to improve user experience, discovery spam emails [6] to ensure the safety of customer services, etc.

Due to the large scale of graph streams and limited memory space, it is almost infeasible to calculate the real number of triangles by storing the entire graph streams. On the one hand, it will consume much more time on the communications between memory and secondary storage by storing the graph streams data into secondary memory. On the other hand, the fully dynamic characteristics of evolving graph streams also induces the difficulties of accurately counting global and local triangles within limited memory space. In the past few decades, there have been plenty of research [7]–[11] on stream sampling methods, which sample a small population of graph streams to compute the number of triangles as accurate as possible.

According to whether memory budget consumed by various sampling methods is fixed or not, we can classify the sampling methods into two categories: Reservoir-based sample and Bernoulli-based sample. The former will initially set a fixed memory budget, which stores the uniform sample chosen from graph streams. However, its sampling ratio monotonously decreases along the growing graph streams after the number of input edges exceeds memory budget. On the contrary, each member of the population has the same probability to be chosen and the inclusion variables are jointly independent in the Bernoulli-based sample. Thus, the memory budget used by Bernoulli-based sampling can vary in principle from 0 to the entire population size, which may exceed the available memory space and cannot be bounded to an expected value.

So far, existing streaming algorithms for triangle counting either fail to maintain a stable sampling ratio or controllable memory budget with the growth of graph streams. Taking full advantage of the characteristics of bounded sampling ratio and efficient utilization of memory budget, we propose a novel sampling method, called bounded-sampling-ratio sample (BSR-Sample), to maintain the sampling ratio greater than or equal to a specified threshold, when there is enough memory budget. The main contributions of this paper are as follows:

- Propose a novel and general sampling method, BSR-Sample, to keep the sampling-ratio great than or equal to a specified sampling ratio threshold, when available memory is enough large. To the best of our knowledge, BSR-Sample is the first attempt to adaptively increment memory budget with the continuous growth of graph streams. The highlight of this method is that we are able to maintain a bounded-sampling-ratio, without requiring any prior knowledge about the scale of graph streams.
- BSR-TC, a streaming algorithm for both global and local triangle counting over evolving graph streams, is proposed based on BSR-Sample. Compared with previous work for triangle counting, it is capable of discovering more triangles and attain higher accuracy by adaptively

DOI reference number: 10.18293/SEKE2021-012

resizing memory budget upwards.

- Experimental results performed on real-world datasets show that BSR-TC can obtain more accurate estimation than the state-of-the-art sampling methods, with the growth of graph streams. Meantime, BSR-TC can keep stable results for both global and local triangle counting with the same specified threshold, regardless of differently initial memory budget.

The rest of this paper is organized as follows. In Section II, we review the related works. Then, we discuss the motivation of our work in Section III. In Section IV, BSR-Sample and BSR-TC are proposed and introduced in detail. In Section V, experiments are conducted using real-world datasets. In Section VI, we conclude our work.

II. Related Work

In this section, we mainly introduce two categories of sampling methods, Bernoulli-based sample, and Reservoir-based sample, for triangle counting over graph streams. They are distinguished by whether consumed memory budget is fixed or not, as illustrated in Table I.

A. Reservoir-based sample

These sampling methods set a fixed memory budget, which is of importance considering the limited memory space. Vitter discussed optimized sampling algorithms in details based on the naive reservoir sampling method, and these optimizations improved the speed by an order of magnitude [12]. The main idea is to skip over a number of records rather than process all the records, reducing the called number of random number generators. However, the optimized Reservoir-based sampling methods are not suited for triangle counting, because they fail to update the estimations for every edge. Gemulla et al. further proposed a novel sampling method based the naive reservoir sampling, called Random Pairing (RP), which handled both edge insertions and deletions for graph streams by the strategy of using future inserted edges to compensate for previous deletions [13]. [14] utilized temporal locality, where future edges were more likely to form triangles with recent edges than older ones, to improve the estimation accuracy. TRI-EST was the first one to estimate triangles in fully-dynamic graph streams, involving both edge insertions and deletions by Reservoir-based sampling methods and its variants [15]. [16] proposed a family of algorithms for global and local triangle counting, called ThinkD, to further improve TRIEST by leveraging unsampled edges to update the estimations of triangles.

B. Bernoulli-based sample

The Bernoulli-based sampling method is relatively simple and efficient for it just needs an initial sampling probability to sample edges over graph streams. Therefore, this sampling method attracts considerable attention to count triangles in evolving graph streams. Ahmed et al. proposed a general sampling framework called graph sample and hold (gSH) for big-graph analytics by one single pass [17]. The gSH utilizes different sampling probabilities based on the graph properties of interest, e.g. gSH(p, q) samples the current arriving edge with probability p when it depends on previously sampled edges, otherwise holds the edge with probability q. Later Ahmed et al. proposed a new framework called graph priority sampling (GPS) for sequentially sampling over evolving graph streams [18]. Two estimation approaches are proposed to attain unbiased estimation of various graph properties, which are post-stream estimation and in-stream estimation. Lim et al. proposed a memory-efficient and accurate method for local triangle estimation over graph streams, called MASCOT [19]. It achieves best performance of both accuracy and memory efficiency of local triangle counting, by the means of "unconditional counting before sampling".

TABLE I: Comparison of sampling methods. Note that the number of current edges is more than initial reservoir size and available memory space is enough large for the sake of simplicity.

Sampling Method	Sampling Ratio	Memory Budget
BSR-Sample (Proposed)	$\geq Threshold$	$Adaptive$
Reservoir-based sample	$Decreased$	$Fixed$
Bernoulli-based sample	$Fixed$	$Increased$

III. Motivation

The sampling ratio of Reservoir-based sample will monotonically decrease after the size of arriving edges exceeds the capacity of memory budget, which inevitably affect the estimation results, as shown by Figure 1. The memory budget of Bernoulli-based sample is not fixed or monotonically increases over the evolving datasets. Figure 2 shows that the sample size of Bernoulli-based sample fluctuates around the real value. Therefore, for the Bernoulli-based sample, it is difficult to allocate appropriate memory space for triangle counting over evolving graph streams.

Considering the characteristics of graph streams, we obtain observations which pose huge challenge for accurate triangle counting.

Observations:

• In real application scenarios, the scale of graph streams is unknown in advance.

• The evolving graph streams usually grow upwards as a whole.

• Memory space is limited to store all edges of graph streams.

These observations above lead to new challenges for global and local triangle counting over evolving graph streams, as described below.

Proposed problems:

• How to maintain an appropriate sampling ratio and how much memory space to allocate for accurate global and local triangle counting, without knowing any prior knowledge of evolving graph streams?

Goals:

• Maintain a bounded-sampling-ratio to achieve accurate and stable estimation of global and local triangle counting.

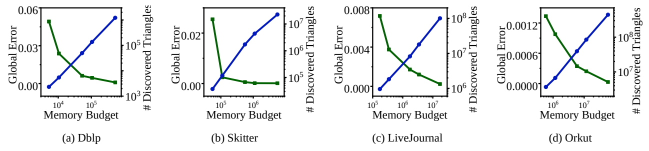


Fig. 1: The *blue line* represents the number of discovered triangles, and the *green line* represents global error. For Reservoir-based sampling methods, both the number of discovered triangles and the accuracy of global triangles are positively correlated with memory budget.

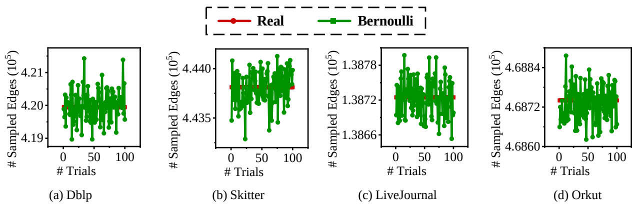


Fig. 2: The sample size of Bernoulli-based methods fluctuates around the real one with sampling probability p=0.4.

• Control Consumed memory budget to improve the efficiency of memory usage.

• Develop single-pass streaming algorithm to avoid multi-repeated and redundant operations.

• And no need to access slower secondary storage.

In general, Reservoir-based sample may induce the sampling ratio to monotonically decrease, which will damage the accuracy of estimations over evolving graph streams. Bernoulli-based sample cannot accurately calculate the size of memory budget. Therefore, we propose a novel Reservoir-based sample method, called BSR-Sample, which keeps a bounded sampling ratio and takes advantage of both methods above. BSR-Sample can maintain the sampling ratio greater than or equal to a given threshold and enable the memory budget to adaptively increase under control within available memory space. The comparison of the three sampling methods is illustrated in Table I.

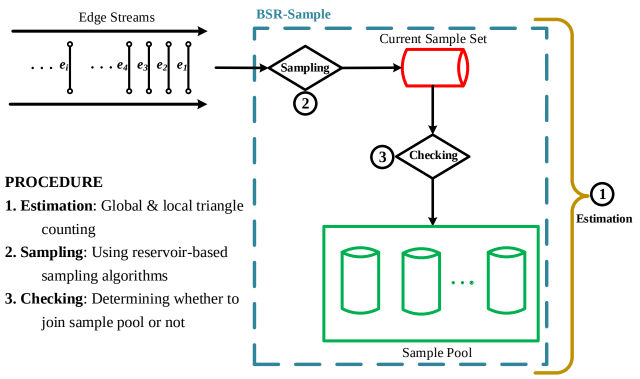
IV. Design and Analysis

In this section, we firstly introduce the overview of BSR-Sample and BSR-TC. Then, we show the implementation of these algorithms in detail.

A. Overview

Our proposed BSR-Sample always maintains the sampling ratio greater than or equal to a specified sampling-ratio threshold by leveraging multi-sets of Reservoir-based sample. Then, we propose BSR-TC using BSR-Sample method for triangle counting. As illustrated in Figure 3, BSR-Sample takes edges from graph streams as input, and outputs a set of sampled edges to BSR-TC for both global and local triangle counting.

• **Estimation.** BSR-TC firstly estimates the number of global and local triangles for each arriving edge from the evolving graph streams, rather than samples them. Here, we



Edge Streams
BSR-Sample
Current Sample Set
Sampling
Checking
Estimation
Sample Pool
PROCEDURE
1. Estimation: Global & local triangle counting
2. Sampling: Using reservoir-based sampling algorithms
3. Checking: Determining whether to join sample pool or not

Fig. 3: The framework of BSR-Sample & BSR-TC.

call this mechanism as "first counting, then sampling", which improves the estimation accuracy by leveraging more edges to participate in statistics analysis.

• **Sampling.** The whole sampling set of BSR-Sample is divided into two parts: the current sampling set and the sampling pool. The former is used to maintain dynamically updated edges, which are sampled based on the naive Reservoir-based sample. The sampling ratio of the current sampling set is always greater than the specified threshold. BSR-Sample will remove the sampled edges into sampling pool once it equals the threshold, and allocates new memory budget as the current sampling set.

• **Checking.** In this procedure, BSR-Sample determines whether to enable the current sampling set join into the sampling pool. When the sampling ratio of the current sampling set equals the specified sampling-ratio threshold, BSR-Sample will remove the entire current sampling set into sampling pool, and then create a new sampling set to substitute for current sampling set. Through adaptively incrementing the current sampling set, BSR-Sample is capable of keeping a bounded-sample-ratio when the available memory is large enough. And BSR-Sample can ensure the estimation accuracy without requiring any prior knowledge about evolving graph streams.

B. Algorithm Description

Here, we first introduce the sampling method of proposed BSR-Sample. Then, we analyze how to estimate global and local triangles by BSR-TC over evolving graph streams. We utilize the naive Reservoir-based sample for the current sampling set to sample edges.

Here, Θ is the totally available memory space. $\mathcal{S}_c$ is the current samping set. $\mathcal{S}_p$ is the samping pool, and $\mathcal{S}_p^{(i)}$ is the ith single sampling set of $\mathcal{S}_p$. $\mathcal{S} = \mathcal{S}_c \cap \mathcal{S}_p$, $\mathcal{N}_u^{\mathcal{S}}$ is the set of neighbors of the node u in $\mathcal{S}$. $\mathcal{M}$ is the initial memory budget. $\mathcal{R}$ is a specified sampling-ratio threshold. Let $\mathcal{M}/\mathcal{R}$ is $\mathcal{T}$. Note that, to simplify the description, we initialize the current sampling set and each single sampling set in the sampling pool to a same size.

• **CountTriangle (Lines 8-20 of Algorithm 2).** In this function unit, CountTriangle first checks whether each node of the arriving edge(u, v) is contained in $\mathcal{S}$ (lines 9-14),

Algorithm 1: Bounded-Sampling-Ratio Sample (BSR-Sample)

```
Input: (1) ${e^{(1)}, e^{(2)}, \cdots}$: a graph stream;
       (2) $\mathcal{M}$: initial memory budget;
       (3) $\mathcal{R}$: specified sampling ratio threshold.
Output: $\mathcal{S}$: a set of sampled edges.
1  for each new arriving edge $e_t = (u, v)$ do
2      SampleEdge($(u, v), \mathcal{S}_c$).
3      CheckRatio($(u, v), \mathcal{S}_c, \mathcal{S}_p$).
4  end
5  Function SampleEdge($e_t, \mathcal{S}_c$)
6      $t_c \leftarrow t \,\%\, \mathcal{M}$
7      if $t_c \leq \mathcal{M}$ then
8          $\mathcal{S}_c \leftarrow \mathcal{S}_c + \{(u, v)\}$
9      end
10     else if a generated random number $(0, 1) \leq \mathcal{M}/t_c$ then
11         choose a random edge $(m, n)$ from $\mathcal{S}_c$
12         $\mathcal{S}_c \leftarrow \mathcal{S}_c - \{(m, n)\}$
13         $\mathcal{S}_c \leftarrow \mathcal{S}_c + \{(u, v)\}$
14     end
15 End
16 Function CheckRatio($e_t, \mathcal{S}_c, \mathcal{S}_p$)
17     $t_c \leftarrow t \,\%\, \mathcal{M}$
18     if $\mathcal{M}/t_c = \mathcal{R}$ and $\Theta \geq \mathcal{M}$ then
19         remove the current sampling set $\mathcal{S}_c$ into sampling pool $\mathcal{S}_p$
20         create a new $\mathcal{S}_c$ with size $\mathcal{M}$
21         $\Theta \leftarrow \Theta - \mathcal{M}$
22     end
23 End
```

which is the output of BSR-Sample in Algorithm 1. Then, we count the common neighbor $\mathcal{N}^{\mathcal{S}}_{u,v}$ of nodes u and v(line 15). For each node c in $\mathcal{N}^{\mathcal{S}}_{u,v}$, we updates both global and local triangle counting by $1/p_{cuv}$ (lines 16-19). Note that $\mathcal{S}_c$ and $\mathcal{S}^{(i)}_p$ in $\mathcal{S}_p$ are produced by the naive Reservoir-based sample, respectively. Therefore, BSR-TC is unbiased for triangle counting, where the expected value equals the real number of triangles. To compute the probability p_{cuv} that BSR-TC discovers the triangle (c, u, v), we divide discovered triangles into 4 types, depending on the positions ($\mathcal{S}_c$ or $\mathcal{S}_p$) of edges (u, c) and (v, c). When a new edge $e_{(t+1)} = (u, v)$ arrives and forms a triangle with a node c, p_{cuv} is calculated by following formula (1).

$$p_{cuv} = \begin{cases} min\{1, \frac{\mathcal{M}}{t_c} \times \frac{\mathcal{M}-1}{t_c-1}\}, \{(u,c),(v,c)\} \in \mathcal{S}_c \\ min\{1, \frac{\mathcal{M}}{t_c}\} \times \mathcal{R}, \{(u,c),(v,c)\} \in \mathcal{S}_c \cup \mathcal{S}^{(i)}_p \\ \mathcal{R} \times \mathcal{R}, \{(u,c),(v,c)\} \in \mathcal{S}^{(i)}_p \\ \mathcal{R} \times \frac{\mathcal{M}-1}{\mathcal{T}-1}, \{(u,c),(v,c)\} \in \mathcal{S}^{(i)}_p \cup \mathcal{S}^{(j)}_p, \forall i \neq j \end{cases} \tag{1}$$

Algorithm 2: BSR-Sample for Triangle Counting (BSR-TC)

```
Input: $\mathcal{S}$: set of sampled edges.
Output: (1) $\Delta$: global triangle counting;
        (2) $\Delta_u$: local triangle counting for node $u$.
1  $\Delta \leftarrow 0$
2  for each new arriving edge $e_t = (u, v)$ do
3      CountTriangle($(u, v)$).
4      SampleEdge($(u, v), \mathcal{S}_c$).
5      CheckRatio($(u, v), \mathcal{S}_c, \mathcal{S}_p$).
6  end
7  Function CountTriangle($(u, v)$)
8      if $u \notin \mathcal{V}$ then
9          $\mathcal{V} \leftarrow \mathcal{V} \cup \{u\}$ and $\Delta_u \leftarrow 0$
10     end
11     if $v \notin \mathcal{V}$ then
12         $\mathcal{V} \leftarrow \mathcal{V} \cup \{v\}$ and $\Delta_v \leftarrow 0$
13     end
14     $\mathcal{N}^{\mathcal{S}}_{u,v} \leftarrow \mathcal{N}^{\mathcal{S}}_u \cap \mathcal{N}^{\mathcal{S}}_v$
15     for each $c \in \mathcal{N}^{\mathcal{S}}_{u,v}$ do
16         $\Delta \leftarrow \Delta + 1/p_{cuv}$;   $\Delta_c \leftarrow \Delta_c + 1/p_{cuv}$;
17         $\Delta_u \leftarrow \Delta_u + 1/p_{cuv}$;   $\Delta_v \leftarrow \Delta_v + 1/p_{cuv}$;
18     end
19 End
```

- **SampleEdge (Lines 5-15 of Algorithm 1).** This function determines whether to sample the arriving edges from graph streams. We first calculate the order of edge streams occurring in the current sampling set $\mathcal{S}_c$ (line 6). For $\mathcal{S}_c$, we adopt the naive Reservoir-based method to sample the arriving edge with probability $min\{1, \mathcal{M}/t_c\}$.
- **CheckRatio (Lines 16-23 of Algorithm 1).** Once the sampling ratio of $\mathcal{S}_c$ reaches a specified threshold $\mathcal{R}$ and there is enough memory space, CheckRatio will remove $\mathcal{S}_c$ into $\mathcal{S}_p$ (line 18-19). Meanwhile, new memory budget is allocated to restart a new round Reservoir-based sampling from scratch. Thus, BSR-Sample can keep the sampling ratio greater than or equal to $\mathcal{R}$ without knowing any knowledge about evolving graph streams.

V. Experiments

We show that BSR-TC suffices to provide accurate estimation for both global and local triangle counting, without requiring any knowledge about graph streams. Compared with the state-of-the-art streaming algorithms (ThinkDAcc, MASCOT and WRS), BSR-TC always maintains a bounded sampling-ratio to discover more triangles along with the continuous growth of graph streams, and so as to obtains more accurate results and efficient memory usage. Therefore, BSR-TC has adaptive characteristics for triangle counting over evolving graph streams.

A. Experimental Setup

We perform experiments on a server with Intel Xeon Gold 6148 processors and 64-bit Red Hat Linux OS. Each experi-

TABLE II: Summary of the real-world graph streams used in our experiments.

Name	# Nodes	# Edges	Summary
Dblp	$317,080$	$1,049,866$	Collaboration network
Skitter	$1,696,415$	$11,095,298$	Internet topology graph
LiveJournal	$3,997,962$	$34,681,189$	Friendship network
Orkut	$3,072,441$	$117,185,083$	Online social network

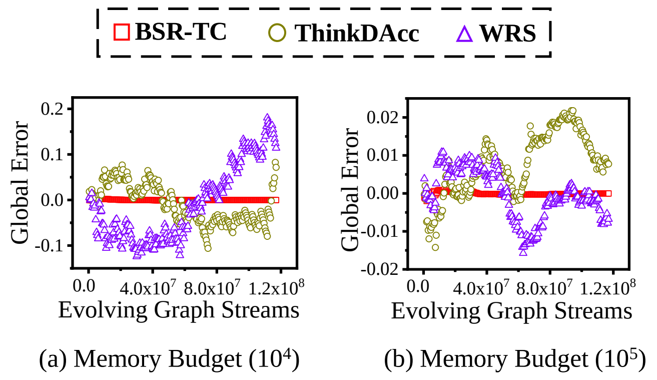


Fig. 4: **Scalability**. BSR-TC is always more accurate and stable than ThinkDAcc and WRS. Here, We calculate the global error rate every 500,000 edges for evolving graph streams, based on the Orkut dataset.

ment entails 100 runs to guarantee the statistical stability of assessment. All experiments are conducted on the real-world graphs from [20], which are summarized in Table II. Suppose that the scale of these datasets is unknown to simulate real application scenarios.

B. Evaluation Metrics

We use the following metrics to evaluate the accuracy of global and local triangle counting, respectively.

- Global Error. Let $\hat{x}$ be the ground truth of the global triangles, and x be the estimated value of $\hat{x}$. Considering $\hat{x}$ may be equal with 0, we add 1 to both $\hat{x}$ and x. Let c denote the number of runs for each experiment. Then, the global error is

$$\frac{1}{c}\sum\nolimits_{i=1}^{c}\frac{|\hat{x}-x|}{\hat{x}+1}$$

- Local Error. Let $\hat{x}_u$ be the ground truth of the local triangles for each node $u \in \mathcal{V}$, and x_u be the estimated value of $\hat{x}_u$. Considering $\hat{x}_u$ may equal 0, we add 1 to both $\hat{x}_u$ and x_u. Then, the local error is

$$\frac{1}{c}\sum\nolimits_{i=1}^{c}\left\{\frac{1}{|\mathcal{V}|}\sum\nolimits_{u\in\mathcal{V}}\frac{|\hat{x}_u-x_u|}{\hat{x}_u+1}\right\}$$

C. Performance

Since BSR-TC is a first exploration of triangle counting by adaptively resizing memory budget over evolving graph streams, there are no existing streaming algorithms for similar comparisons. For the sake of illustration, we use ThinkDAcc, MASCOT and WRS as the baselines, which are state-of-the-art streaming algorithms for triangle counting. Here, we define

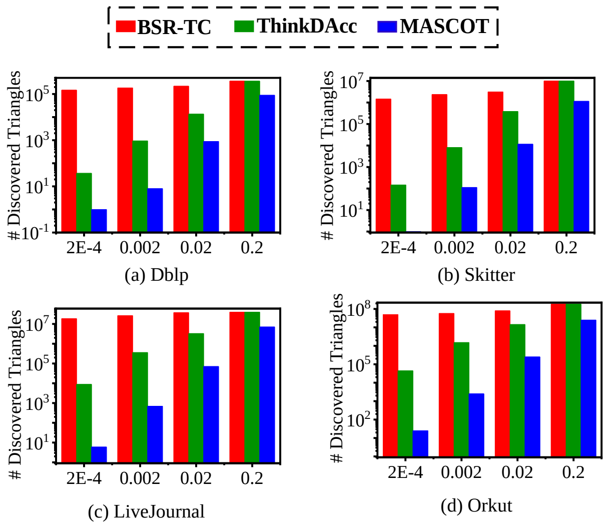


Fig. 5: **Accuracy**. More triangles are discovered by BSR-TC, when $\eta < \mathcal{R}$.

the ratio of initial memory budget against whole graph streams as η, and set the threshold $\mathcal{R}$ to 20%.

- **Scalability (maintain accuracy over evolving streams)**. In real application scenarios, it is challenging to set appropriate parameters, such as memory budget, for triangle counting, because the scale of graph streams is always increasing as time flies. As shown in Figure 4, when initial memory budget is 10^4 edges, the global errors of both ThinkDAcc and WRS fluctuate between -0.15 and 0.2 as graph streams evolves, while BSR-TC is almost always equal to zero. This is because the memory budget of ThinkDAcc and WRS is fixed and is difficult to set appropriately without any knowledge about the scale of graph streams. Our proposed BSR-TC can keep stable and high accuracy by the adaptively resampling method over evolving graph streams.
- **Accuracy (regardless of small η)**. Figure 5 shows BSR-TC discovers more triangles than ThinkDAcc and MASCOT by adaptively resampling method, when η is less than $\mathcal{R}$. By sampling the first edges in memory budget with probability 1, ThinkDAcc always discovers more triangles than MASCOT. Figure 6 depicts that BSR-TC achieves accuracy of at least 99.8% for global triangles and 60.0% for local triangles, respectively, when $\eta \geq 0.002\%$ and $\mathcal{R} = 20\%$. Under this condition, BSR-TC gains accuracy of $100\times$ for global triangles than ThinkDAcc. Therefore, our proposed BSR-TC can maintain high accuracy by adaptively incrementing memory budget to maintain a bounded sampling ratio $\mathcal{R}$ over the growth of graph streams, even though the initial η is small.

VI. Conclusions

We propose a single-pass and bounded-sampling-ratio method, BSR-Sample, by adaptively resizing memory budget under control without requiring any prior knowledge about graph streams. BSR-Sample allocates new memory budget to restart a new round of sampling based on standard reservoir

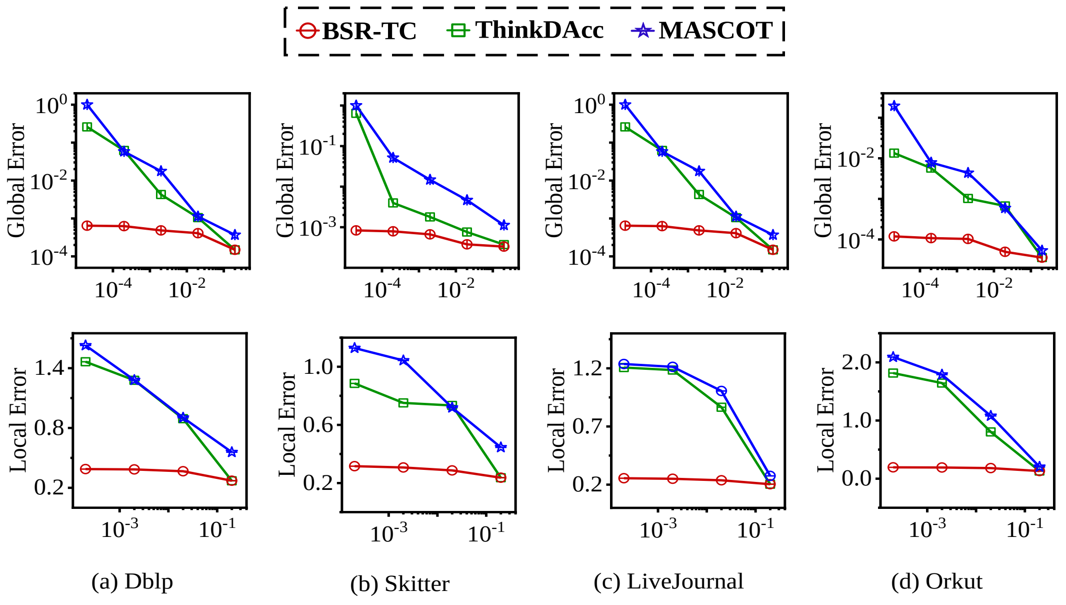


Fig. 6: **Accuracy**. BSR-TC is more accurate than state-of-the-art streaming algorithms for global and local triangle counting. Here, the $\mathcal{X}$*-axis* denotes η.

sampling, when current sampling ratio is less than a given threshold. Then, by the mechanism as “first counting, then sampling”, we propose BSR-TC for global and local triangle counting based on BSR-Sample. To the best of our knowledge, BSR-TC is the first attempt to adaptively resize memory budget over evolving graph streams. Compared to state-of-the-art streaming algorithms, BSR-TC can obtain more accurate and stable estimation of triangles over evolving graph streams.

VII. Acknowledgment

This work was supported by the National Natural Science of China (11904370, 61872335, 61732018, 61672499), the Innovation Project of the State Key Laboratory of Computer Architecture (CARCH4509), and the Open Project Program of the State Key Laboratory of Mathematical Engineering and Advanced Computing (2019A07).

References

[1] N. K. Ahmed, J. Neville, and R. Kompella, “Network sampling: From static to streaming graphs,” *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 8, no. 2, pp. 1–56, 2013.
[2] S. Wasserman, K. Faust *et al.*, “Social network analysis: Methods and applications,” 1994.
[3] A. McGregor, “Graph stream algorithms: a survey,” *ACM SIGMOD Record*, vol. 43, no. 1, pp. 9–20, 2014.
[4] A. Zakrzewska and D. A. Bader, “A dynamic algorithm for local community detection in graphs,” in *2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. IEEE, 2015, pp. 559–564.
[5] T. Opsahl and P. Panzarasa, “Clustering in weighted networks,” *Social networks*, vol. 31, no. 2, pp. 155–163, 2009.
[6] L. Becchetti, P. Boldi, C. Castillo, and A. Gionis, “Efficient algorithms for large-scale local triangle counting,” *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 4, no. 3, pp. 1–28, 2010.
[7] K. Kutzkov and R. Pagh, “Triangle counting in dynamic graph streams,” in *Scandinavian Workshop on Algorithm Theory*. Springer, 2014, pp. 306–318.
[8] C. C. Aggarwal, “On biased reservoir sampling in the presence of stream evolution,” in *Proceedings of the 32nd international conference on Very large data bases*, 2006, pp. 607–618.
[9] C. E. Tsourakakis, “Fast counting of triangles in large real networks without counting: Algorithms and laws,” in *2008 Eighth IEEE International Conference on Data Mining*. IEEE, 2008, pp. 608–617.
[10] B. Wu, K. Yi, and Z. Li, “Counting triangles in large graphs by random sampling,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 8, pp. 2013–2026, 2016.
[11] P. Wang, Y. Qi, Y. Sun, X. Zhang, J. Tao, and X. Guan, “Approximately counting triangles in large graph streams including edge duplicates with a fixed memory usage,” *Proceedings of the VLDB Endowment*, vol. 11, no. 2, pp. 162–175, 2017.
[12] J. S. Vitter, “Random sampling with a reservoir,” *ACM Transactions on Mathematical Software (TOMS)*, vol. 11, no. 1, pp. 37–57, 1985.
[13] R. Gemulla, W. Lehner, and P. J. Haas, “Maintaining bounded-size sample synopses of evolving datasets,” *The VLDB Journal*, vol. 17, no. 2, pp. 173–201, 2008.
[14] K. Shin, “Wrs: Waiting room sampling for accurate triangle counting in real graph streams,” in *2017 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2017, pp. 1087–1092.
[15] L. D. Stefani, A. Epasto, M. Riondato, and E. Upfal, “Triest: Counting local and global triangles in fully dynamic streams with fixed memory size,” *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 11, no. 4, pp. 1–50, 2017.
[16] K. Shin, S. Oh, J. Kim, B. Hooi, and C. Faloutsos, “Fast, accurate and provable triangle counting in fully dynamic graph streams,” *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 14, no. 2, pp. 1–39, 2020.
[17] 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*, 2014, pp. 1446–1455.
[18] N. K. Ahmed, N. Duffield, T. Willke, and R. A. Rossi, “On sampling from massive graph streams,” *arXiv preprint arXiv:1703.02625*, 2017.
[19] 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*, 2015, pp. 685–694.
[20] J. Leskovec and A. Krevl, “SNAP Datasets: Stanford large network dataset collection,” http://snap.stanford.edu/data, Jun. 2014.