DiSLR: Distributed Sampling with Limited Redundancy For Triangle Counting in Graph Streams

Kijung Shin[1], Euiwoong Lee[1], Jinoh Oh[2], Mohammad Hammoud[3], Christos Faloutsos[1]
[1]Carnegie Mellon University, Pittsburgh, PA, USA, {kijungs,euiwoonl,christos}@cs.cmu.edu
[2]Adobe Systems, San Jose, CA, USA, joh@adobe.com
[3]Carnegie Mellon University in Qatar, Doha, Qatar, mhhamoud@cmu.edu

ABSTRACT

Given a web-scale graph that grows over time, how should its edges be stored and processed on multiple machines for rapid and accurate estimation of the count of triangles? The count of triangles (i.e., cliques of size three) has proven useful in many applications, including anomaly detection, community detection, and link recommendation. For triangle counting in large and dynamic graphs, recent work has focused largely on streaming algorithms and distributed algorithms. To achieve the advantages of both approaches, we propose DiSLR, a distributed streaming algorithm that estimates the counts of global triangles and local triangles associated with each node. Making one pass over the input stream, DiSLR carefully processes and stores the edges across multiple machines so that the redundant use of computational and storage resources is minimized. Compared to its best competitors, DiSLR is **(a) Accurate:** giving up to *39× smaller* estimation error, **(b) Fast**: up to *10.4× faster*, scaling linearly with the number of edges in the input stream, and **(c) Theoretically sound**: yielding unbiased estimates with variances decreasing faster as the number of machines is scaled up.

1 INTRODUCTION

Given a graph stream, how can we utilize multiple machines for rapidly and accurately estimating the count of triangles? How should we process and sample the edges across the machines to minimize the redundant use of computational and storage resources?

The count of triangles (i.e., cliques of size three) is a computationally expensive but important graph statistic that has proven useful in diverse areas. For example, the counts of global triangles (i.e., all triangles) and local triangles (i.e., triangles associated with each node) lie at the heart of many crucial concepts in social network analysis and graph theory, including the transitive ratio [14], local clustering coefficients [27], social balance [26], and trusses [7]. The global and local triangle counts have also been used in many data mining and database applications, including link recommendation [10, 23], anomaly detection [13], spam detection [6], dense subgraph mining [25], and query optimization [5].

For triangle counting in real-world graphs, many of which are large and evolving with new edges, recent work has focused largely on streaming algorithms. Given a graph stream, which is a sequence of edges that may not fit in the underlying storage, streaming algorithms estimate the count of triangles while making a single pass over the stream. Streaming algorithms maintain and gradually update their estimates as each edge is received rather than operating on the entire graph. Thus, they are also appropriate for dynamic graphs whose edges are received over time. As a result, a number of streaming algorithms for triangle counting (e.g., [2, 3, 9, 12, 13, 16–18]) have been proposed in recent years.

DOI: 10.1145/nnnnnnn.nnnnnnn

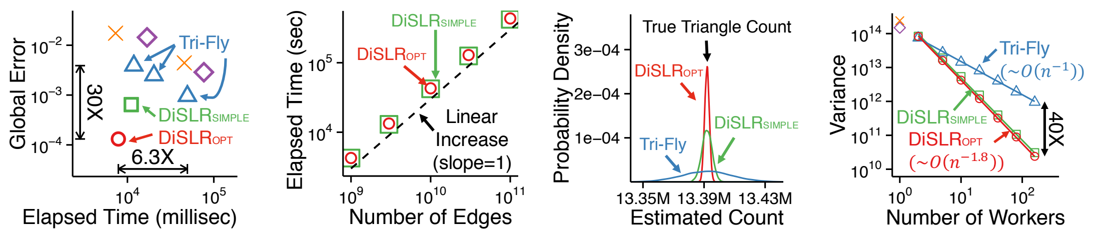
Table 1: Comparison of triangle counting algorithms. Our proposed DiSLR algorithm satisfies every criterion, significantly outperforming [18].

	[22, 24]	[6]	[8, 15, 20]	[4]	[2, 11, 17, 21]	[9, 12, 13]	**DiSLR & [18]**
Single-Pass Stream Processing					✓	✓	✓
Approximation for Large Graphs	✓	✓		✓	✓	✓	✓
Global & Local Triangle Counting		✓	✓		✓		✓
Larger Data with More Machines			✓	✓			✓
More Accurate with More Machines				✓			✓

Another popular approach is to extend triangle counting algorithms to distributed settings, including distributed-memory [4] and MapReduce [8, 15, 20] settings, so that we can utilize computational and storage resources of multiple machines for speed and scalability. However, unlike streaming algorithms, these distributed algorithms require all edges to be given at once. Thus, they are not applicable to dynamic graphs, whose edges are received over time, or graphs that are too large to fit in the underlying storage.

Can we have the best of both worlds? Can we utilize multiple machines for rapid and accurate triangle counting in a graph stream? To the best of our knowledge, the only distributed streaming algorithm that pursues this goal is Tri-Fly [18], as shown in Table 1. In Tri-Fly, every edge is broadcast to every machine that runs a state-of-the-art single-machine streaming algorithm independently. The final estimates of Tri-Fly are the averages of the estimates provided by all the machines. However, Tri-Fly is not satisfactory in terms of speed and accuracy since incurs a highly redundant use of computational and storage resources.

In this work, we propose DiSLR (**Di**stributed **S**ampling with **L**imited **R**edundancy), a distributed streaming algorithm that estimates the counts of global and local triangles. DiSLR gives the advantages of both streaming and distributed algorithms, significantly outperforming Tri-Fly, as shown in Figure 1. DiSLR minimizes the redundant use of computational and storage resources. Specifically, it carefully processes and samples edges across distributed machines so that each edge is stored in at most two machines and each



(a) Speed and Accuracy (b) Scalability (c) Unbiasedness (Theorem 5.1) (d) Variance Decrease (Theorem 5.4)

Figure 1: <u>DiSLR is fast, accurate, scalable, and theoretically sound.</u> (a) DiSLR is faster and more accurate than its competitors. (b) The running time of DiSLR is linear to the number of edges in the input stream. (c)-(d) DiSLR gives unbiased estimates with variances dropping rapidly as we use more machines (Theorems 5.1 and 5.4). See Section 6 for the experimental settings.

triangle is counted by at most one machine. We theoretically and empirically demonstrate that DiSLR has the following advantages:

- **Accurate**: DiSLR yields up to *30*× and *39*× *smaller estimation errors* for global and local triangle counts, respectively, than its competitors with similar speeds (Figure 1(a)).
- **Fast**: DiSLR scales linearly with the number of edges in the input stream, and it is up to *10.4*× *faster* than competitors while giving more accurate estimates (Figures 1(a) and 1(b)).
- **Theoretically Sound:** DiSLR gives unbiased estimates (Figure 1(c) and Theorem 5.1) with variances dropping rapidly as the number of machines is scaled up (Figure1(d) and Theorem 5.4).

In Section 2, we review related work. In Section 3, we give notations and the problem definition. We describe our proposed algorithm DiSLR in Section 4 and give theoretically analyses of it in Section 5. After presenting experimental results in Section 6, we draw conclusions in Section 7.

2 RELATED WORK

We review related work on streaming algorithms and distributed algorithms for triangle counting. See Table 1 for a summary.

2.1 Single-Machine Streaming Algorithms.

Most streaming algorithms for triangle counting employ sampling for estimation with limited storage.

Counting global triangles. Tsourakakis et al. [24] proposed sampling each edge with equal probability and then estimating the count of global triangles from that in the sampled graph. For better accuracy, Jha et al. [11] and Pavan et al. [17] proposed sampling wedges (i.e., paths of length two) in addition to edges; and Ahmed et al. [2, 3] proposed sampling edges with different probabilities depending on the counts of adjacent sampled edges and incident triangles. Pavan et al. [21] parallelized their algorithm [17] for a shared-memory setting. However, this parallelization is applicable only when edges arrive in batches rather than one by one.

Counting local triangles. For local triangle counting in a graph stream, Kutzkov and Pagh [12] proposed coloring nodes and sampling edges whose endpoints have the same color. Lim and Kang [13] proposed MASCOT, which uses simple uniform edge sampling but updates its estimates whenever an edge arrives even if it is not sampled. De Stefani et al. [9] proposed TRIEST$_{\text{IMPR}}$, which uses reservoir sampling to maintain as many sample edges as storage allows. Our algorithm adapts TRIEST$_{\text{IMPR}}$ for triangle counting within each machine since it estimates both global and local triangle counts most accurately without any parameter, However, any single-machine streaming algorithm can be used instead, as explored further in Section F of the supplementary document [1].

2.2 Distributed Batch Algorithms.

Cohen [8] proposed the first triangle counting algorithm on MAPREDUCE, which directly parallelizes a serial algorithm. Suri and Sergei [20], Park et al. [15], and Shaikh et al. [4] proposed dividing the input graph into overlapping subgraphs and assigning them to multiple machines, which count the triangles in the assigned subgraphs in parallel, in MAPREDUCE [15, 20] and distributed-memory [4] settings. These distributed algorithms are for exact triangle counting in static graphs, all of whose edges are given at once. They are not applicable to streaming settings where edges are received over time and all edges may not fit in the underlying storage.

2.3 Distributed Streaming Algorithms.

Distributed sampling algorithms for triangle counting were first discussed by Pavan et al. [16] to handle multiple sources. Their goal, however, was to reduce communication costs while giving the same estimation of their single-machine streaming algorithm [17]. Thus, using more machines, which are one per source, neither improves the speed or accuracy of the estimation. Our goal of using multiple machines for rapid and accurate estimation was pursued by Shin et al. [18]. In their proposed algorithm TRI-FLY, every edge is broadcast to every worker that independently runs TRIEST$_{\text{IMPR}}$. The workers send their estimates to the aggregators, which give the final estimates by averaging the received estimates. Although TRI-FLY gives unbiased estimates whose variances decrease inversely proportional to the number of workers, it incurs a highly redundant use of computational and storage resources. Specifically, in the worst case, each edge is replicated and stored in every worker, and each triangle is counted by every worker. Due to this redundancy, no matter how many workers are used, TRI-FLY cannot guarantee exact triangle counts if the input stream does not fit in a single machine. Our proposed algorithm DiSLR significantly improves upon TRI-FLY in terms of speed and accuracy by limiting the redundancy in storage and computation.

3 NOTATIONS AND PROBLEM DEFINITION

We introduce notations and give a formal problem definition. We list the frequently-used symbols in Table 2

3.1 Notations

Consider a graph stream $(e^{(1)}, e^{(2)}, \ldots)$, where $e^{(t)}$ denotes the undirected non-parallel non-self edge that arrives at time $t \in \{1, 2, \ldots\}$. Then, let $\mathcal{G}^{(t)} = (\mathcal{V}^{(t)}, \mathcal{E}^{(t)})$ be the graph composed of the nodes and edges arriving at time t or earlier.[1] We use the unordered

[1] i.e., $\mathcal{V}^{(0)} := \emptyset$, $\mathcal{E}^{(0)} := \emptyset$, $\mathcal{V}^{(t+1)} := \mathcal{V}^{(t)} \cup e^{(t+1)}$, and $\mathcal{E}^{(t+1)} := \mathcal{E}^{(t)} \cup \{e^{(t+1)}\}$.

Table 2: **Table of frequently-used symbols.**

Symbol	Definition
Notations for Graph Streams (defined in Section 3.1)	
$(e^{(1)}, e^{(2)}, \ldots)$	input graph stream
$e^{(t)}$	edge that arrives at time t
$\{u, v\}$	edge between nodes u and v
t_{uv}	arrival time of edge $\{u, v\}$
$\{u, v, w\}$	triangle with nodes u, v, and w
$\mathcal{G}^{(t)} = (\mathcal{V}^{(t)}, \mathcal{E}^{(t)})$	graph at time t
$\mathcal{T}^{(t)}$	set of global triangles in $\mathcal{G}^{(t)}$
$\mathcal{T}^{(t)}[u]$	set of local triangles associated with node u in $\mathcal{G}^{(t)}$
Notations for Algorithms (defined in Section 4.1)	
$\mathcal{W}$	set of workers
k	maximum number of edges stored in each worker
l_i	load of worker i
$\bar{c}$	estimate of global triangle count
$c[u]$	estimate of local triangle count of node u
$f : \mathcal{V} \to \mathcal{W}$	function assigning nodes to workers
θ	tolerance for load difference
Notations for Analyses (defined in Section 5.1)	
$p^{(t)}$	number of Type 1 triangle pairs in $\mathcal{G}^{(t)}$
$q^{(t)}$	number of Type 2 triangle pairs in $\mathcal{G}^{(t)}$

pair $\{u, v\} \in \mathcal{E}^{(t)}$ to indicate the edge between two distinct nodes $u, v \in \mathcal{V}^{(t)}$. We denote the arrival time of each edge $\{u, v\}$ by t_{uv}.[2] We use the unordered triple $\{u, v, w\}$ to indicate the triangle (i.e., cliques of size three) composed of three distinct nodes $u, v, w \in \mathcal{V}^{(t)}$. We let $\mathcal{T}^{(t)}$ be the set of *global triangles* in $\mathcal{G}^{(t)}$ (i.e., all triangles in $\mathcal{G}^{(t)}$), and for each node $u \in \mathcal{V}^{(t)}$, we let $\mathcal{T}^{(t)}[u] \in \mathcal{T}^{(t)}$ be the *local triangles of* u in $\mathcal{G}^{(t)}$ (i.e., triangles associated with u in $\mathcal{G}^{(t)}$).

3.2 Problem Definition

We consider Problem 1, the distributed estimation of the counts of global and local triangles in a graph stream.

Problem 1 (Distributed Estimation of the Global and Local Triangle Counts in a Graph Stream).

(1) **Given:** *a graph stream* $(e^{(1)}, e^{(2)}, \ldots)$*, and n distributed storages, in each of which up to* k (≥ 2) *edges can be stored.*
(2) **to Minimize:** *the estimation error of the global triangle count* $|\mathcal{T}^{(t)}|$ *and local triangle counts* $\{(u, |\mathcal{T}^{(t)}[u]|)\}_{u \in \mathcal{V}^{(t)}}$ *for each time* $t \in \{1, 2, \ldots\}$.
(3) **Subject to:** *the following realistic conditions:*
  C1 **knowledge free:** *no prior knowledge of the input graph stream (e.g., the size of the stream) is available.*
  C2 **shared nothing environment:** *data stored in the storage of a machine is not accessible by the other machines.*
  C3 **one pass:** *edges are accessed one by one in their arrival order. Past edges are not accessible by a machine unless they are stored in the given storage of the machine.*

Instead of minimizing a specific measure of estimation error, we use a general approach of simultaneously reducing bias and variance to reduce different measures of estimation error robustly.

[2]i.e., $e^{(t)} = \{u, v\} \equiv t_{uv} = t$.

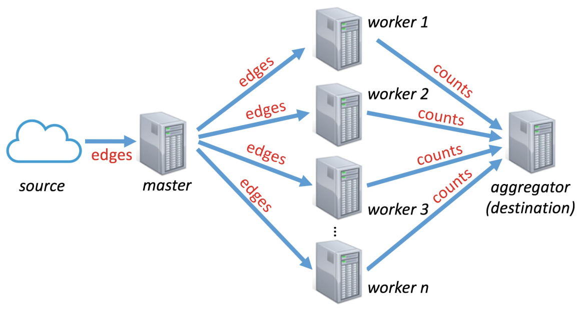


Figure 2: Roles of machines and the flow of data in DiSLR. Extensions with multiple sources, masters, and aggregators are discussed in Section 4.6.

4 PROPOSED METHOD: DISLR

We propose DiSLR (**Di**stributed **S**ampling with **L**imited **R**edundancy), a distributed streaming algorithm for Problem 1. We first give an overview and the design goals. Then, we present the details of DiSLR. After that, we discuss optimizing node mapping, lazy aggregation, and extensions with multiple masters and aggregators.

4.1 Overview

Figure 2 describes the roles of machines and the flow of data in DiSLR. For simplicity, we assume one source, one master and one aggregator although extensions with multiple of them are trivial, as discussed in Section 4.6. Edges are streamed from the source to the master, which broadcasts or multicasts the edges to the workers. Each worker counts the global and local triangles from the received edges using its local storage, and it sends the counts to the aggregator. Since we assume a shared-nothing environment in Problem 1, each worker cannot access data stored in the other workers. The counts are aggregated in the aggregator, which gives the final estimates of the global and local triangle counts.

4.2 Design Goals

We design DiSLR so that it has the following desirable properties:

P1 **Limited redundancy in storage:** Each edge is stored in at most two workers.
P2 **No redundancy in computation:** Each triangle is counted by at most one worker.
P3 **No definitely missing triangles:** Each triangle is counted with non-zero probability.

P1 is desirable since less redundancy in storage enables us to store more unique edges, which we can estimate triangle counts more accurately from. P2 is of course for speed. P3, which enables us to give unbiased estimates of triangle counts, is what we should not compromise while reducing the redundancy in storage and computation. For example, storing each edge in at most one worker can compromise P3 unless we use only one worker. After discussing the details of DiSLR, we prove that DiSLR satisfies P1, P2, and P3.

4.3 Detailed Algorithm

DiSLR is described in Algorithm 1. We introduce the notations used in it. Then, we explain the master, the workers, and the aggregator. After that, we prove that DiSLR satisfies Properties P1, P2, and P3.

Notations. We use f to indicate a function that maps each node to a worker. We use $\mathcal{W}$ to denote the set of workers and use k to denote the storage budget per worker (i.e., the maximum number of edges that we store in each worker). We let $\mathcal{E}_i$ be the edges currently stored in worker $i \in \mathcal{W}$, and we let $\mathcal{G}_i = (\mathcal{V}_i, \mathcal{E}_i)$ be the graph composed of the edges in $\mathcal{E}_i$. For each node $u \in \mathcal{V}_i$, $\mathcal{N}_i[u]$ denotes the neighboring nodes of u in $\mathcal{G}_i$. Since each worker uses sampling to decide which edges to store, we use l_i to denote the number of edges considered in the sampling so far.[3] Lastly, $\bar{c}$ indicates the estimate of the global triangle count, and for each node u, $c[u]$ indicates the estimate of the local triangle count of u.

Master: The master (lines 1-3) requires a function f that maps each node to a worker. We assume that f is given and discuss it later in Section 4.4. The master sends each edge $\{u, v\}$ to the workers depending on $f(u)$ and $f(v)$ as follows:

- **Case M-1** (line 2): If nodes u and v are assigned to the same worker by f (i.e., $f(u) = f(v)$), then the master sends $\{u, v\}$ only to the worker (i.e., worker $f(u)$).
- **Case M-2** (line 3): Otherwise (i.e., if $f(u) \neq f(v)$), the master sends $\{u, v\}$ to every worker.

Workers: The workers (lines 4-18) start with an empty storage (line 4). Whenever they receive an edge $\{u, v\}$ from the master (line 5), they count the triangles with $\{u, v\}$ in its local storage by calling procedure COUNT (line 6). In COUNT (lines 9-13), each worker $i \in \mathcal{W}$ finds the common neighbors of nodes u and v in the graph $\mathcal{G}_i$, which a graph composed of the edges in $\mathcal{E}_i$ (line 10). Each common neighbor w guarantees that the triangle $\{u, v, w\}$ exists. Therefore, for each such w, the worker sends the increases of the global triangle count and the local triangle counts of nodes u, v, and w to the aggregator (lines 11-13). For each such triangle $\{u, v, w\}$, the amount of increases in the counts is $1/(p_i[uvw])$, where

$$p_i[uvw] := \min\left(1, \frac{k(k-1)}{l_i(l_i-1)}\right) \tag{1}$$

is the probability that worker i discovers the triangle $\{u, v, w\}$ (i.e., the probability that both $\{v, w\}$ and $\{w, u\}$ are in $\mathcal{E}_i$ when $\{u, v\}$ arrives at worker i), as we explain later. This amount of increase is for making the expected amount of increase be exactly 1 $(= p_i[uvw] \times 1/(p_i[uvw]) + (1 - p_i[uvw]) \times 0)$ for each triangle and DiSLR give unbiased estimates, as formalized in Section 5.1.

Unlike procedure COUNT, procedure SAMPLE is called selectively depending on $f(u)$ and $f(v)$ as follows:

- **Case W-1** (line 7): If $f(u) = i$ or $f(v) = i$, then worker i considers storing $\{u, v\}$ in its local storage by calling SAMPLE .
- **Case W-2**: Otherwise (i.e., if $f(u) \neq i$ and $f(v) \neq i$), worker i simply discards $\{u, v\}$ without considering storing it.

In SAMPLE (lines 15-18), each worker $i \in \mathcal{W}$ first increases l_i, the count of edges considered for sampling, by one since the new edge $\{u, v\}$ is being considered. Then, if its local storage has free space (i.e., $|\mathcal{E}_i| < k$), $\{u, v\}$ is stored in the free space and thus added to $\mathcal{E}_i$ (line 16). Otherwise (i.e., $|\mathcal{E}_i| = k$), a random edge in the local storage is replaced with $\{u, v\}$ with probability k/l_i (lines 17-18). This is the standard reservoir sampling. Thus, it is guaranteed that $\mathcal{E}_i$ includes each of the l_i edges with equal probability $\min(1, k/l_i)$ and each pair of the l_i edges with equal probability $p_i[uvw]$ (Eq. (1)).

Aggregator: The aggregator (lines 19-23) applies each received update to the corresponding estimate.

[3]If $l_i > k$, then $l_i \geq |\mathcal{E}_i|$ since not all the considered edges can be stored.

Algorithm 1 D*i*SLR

```
Input: input graph stream: $(e^{(1)}, e^{(2)}, \ldots)$
       storage budget per worker: $k$ $(\geq 2)$
       function assigning nodes to workers: $f$
Output: estimated global triangle count: $\bar{c}$
        estimated local triangle counts: $c[u]$ for each node $u$
Master:
 1: for each edge $\{u, v\}$ from the source do
 2:     if $f(u) = f(v)$ then send $\{u, v\}$ to worker $f(u)$        ▷ Case M-1
 3:     else send $\{u, v\}$ to every worker in $\mathcal{W}$          ▷ Case M-2
Worker (each worker with index $i$):
 4: $\mathcal{E}_i \leftarrow \emptyset$; $l_i \leftarrow 0$
 5: for each edge $\{u, v\}$ from the master do
 6:     COUNT$(u, v)$
 7:     if $f(u) = i$ or $f(v) = i$ then SAMPLE$(u, v)$                ▷ Case W-1
 8: procedure COUNT$(u, v)$:
 9:     $sum \leftarrow 0$
10:     for each node $w \in \mathcal{N}_i[u] \cap \mathcal{N}_i[v]$ do
11:         send $(w, 1/(p_i[uvw]))$ to the aggregator
12:         $sum \leftarrow sum + 1/(p_i[uvw])$      ▷ see Eq. (1) for $p_i[uvw]$
13:     send $(*, sum)$, $(u, sum)$ and $(v, sum)$ to the aggregator
                                 ▷ '$*$' indicates the global triangle count
14: procedure SAMPLE$(u, v)$:
15:     $l_i \leftarrow l_i + 1$.
16:     if $|\mathcal{E}_i| < k$ then $\mathcal{E}_i \leftarrow \mathcal{E}_i \cup \{\{u, v\}\}$
17:     else if a random number in Bernoulli$(k/l_i)$ is 1
18:         replace a random edge in $\mathcal{E}_i$ with $\{u, v\}$
Aggregator:
19: $\bar{c} \leftarrow 0$
20: initialize an empty map $c$ with default value 0
21: for each pair $(u, \delta)$ from the workers do
22:     if $u = *$ then $\bar{c} \leftarrow \bar{c} + \delta$
23:     else $c[u] \leftarrow c[u] + \delta$
```

Properties: DiSLR satisfied Properties P1, P2, and P3, which are our design goals, as stated in Lemma 4.1.

LEMMA 4.1. *Properties P1, P2, and P3 are satisfied in Algorithm 1.*

Proof. First, we prove P1. Each edge $\{u, v\}$ can be stored in a worker only when Case W-1 happens. Since Case W-1 happens in at most two workers (i.e., worker $f(u)$ and worker $f(v)$), $\{u, v\}$ can be stored in at most two workers.

Then, we prove P2 and P3 by showing that, for each triangle, there exists exactly one worker that counts it with non-zero probability. Consider a triangle $\{u, v, w\}$ and assume $\{u, v\}$ is the last edge (i.e., $t_{vw} < t_{uv}$ and $t_{wu} < t_{uv}$) without loss of generality. If $f(u) = f(v)$, all the workers except $f(u)$ cannot count $\{u, v, w\}$ since $\{u, v\}$ is sent only to worker $f(u)$ (Case M-1). Since worker $f(u)$ $(= f(v))$ stores $\{v, w\}$ and $\{w, u\}$ with non-zero probability (Case W-1 happens for both edges), $f(u)$ counts $\{u, v, w\}$ with non-zero probability. If $f(u) \neq f(v)$, although $\{u, v\}$ is sent to every worker (Case M-2), all the workers except $f(w)$ cannot count $\{u, v, w\}$ since they cannot store both $\{v, w\}$ and $\{w, u\}$ (Case W-2 happens for at least one of the edges). Since worker $f(w)$ stores $\{v, w\}$ and $\{w, u\}$ with non-zero probability (Case W-1 happens for both edges), $f(w)$ counts $\{u, v, w\}$ with non-zero probability. Thus, in both cases, there exists exactly one worker that counts $\{u, v, w\}$ with non-zero probability, satisfying P2 and P3. ■

Algorithm 2 Master in DiSLR$_{\text{OPT}}$

Input: input graph stream: $(e^{(1)}, e^{(2)}, \ldots)$
tolerance for load difference: θ (≥ 0) .
Output: send edges to workers

```
Master:
1:  $l_i \leftarrow 0, \forall i \in \mathcal{W}$
2:  for each edge $\{u, v\}$ from the source do
3:      $i^* \leftarrow \arg\min_{i \in \mathcal{W}} l_i$
4:      if $u$ and $v$ have not been assigned to a worker by $f$ then
5:          $f(u) \leftarrow i^*; f(v) \leftarrow i^*$
6:      else if $u$ has not been assigned to a worker by $f$ then
7:          $f(u) \leftarrow f(v)$ if $l_{f(v)} \leq (1+\theta) l_{i^*}$ else $i^*$
8:      else if $v$ has not been assigned to a worker by $f$ then
9:          $f(v) \leftarrow f(u)$ if $l_{f(u)} \leq (1+\theta) l_{i^*}$ else $i^*$
10:     if $f(u) = f(v)$ then                                    ▷ Case M-1
11:         send $\{u, v\}$ to worker $f(u)(= f(v))$
12:         $l_{f(u)} \leftarrow l_{f(u)} + 1$
13:     else                                                     ▷ Case M-2
14:         send $\{u, v\}$ to every worker in $\mathcal{W}$
15:         $l_{f(u)} \leftarrow l_{f(u)} + 1$
16:         $l_{f(v)} \leftarrow l_{f(v)} + 1$
```

4.4 Optimized Node Mapping Function

So far, we have assumed that a function f, which assigns each node to a worker, is given. We discuss how to design f and propose DiSLR$_{\text{OPT}}$, which is DiSLR with our optimized function as f.

Design Goals of f. We say an edge $\{u, v\}$ is assigned to worker i if $f(u) = i$ or $f(v) = i$ and thus $\{u, v\}$ can possibly be stored in worker i. In Algorithm 1, the load l_i of each worker $i \in \mathcal{W}$ denotes the number of edges assigned to worker i. Then, two goals that a desirable function f should meet are as follows:

G1 **Storage:** The redundant use of storage (i.e., the number of edges stored in multiple workers) should be minimized.

G2 **Load Balancing:** A similar number of edges should be assigned to every worker, i.e., $l_i \approx l_j, \forall i, j \in \mathcal{W}$.

However, achieving both goals is non-trivial because the goals compete with each other. For example, if we assign every node to the same worker, then the first goal is achieved since every edge is stored only in the machine. However, this maximizes load imbalance, conflicting with the second goal. Moreover, f should be decided within a single pass without any prior knowledge of the input stream, due to our conditions in Problem 1.

DiSLR$_{\text{OPT}}$ with Optimized f. We propose DiSLR$_{\text{OPT}}$, where the master, described in Algorithm 2, adaptively decides the function f based on current loads of the workers so that the redundancy of storage is minimized within a specified level of load difference.

Recall that, in DiSLR, Case M-1 is preferred over Case M-2 for reducing the redundancy in storage. This is because each edge $\{u, v\}$ is stored in at most one worker in Case M-1 (i.e., $f(u) = f(v)$), while it is stored in at most two workers in Case M-2 (i.e., $f(u) \neq f(v)$). Let worker $i^* \in \mathcal{W}$ be the worker with minimum assigned edges so far (line 3). If an edge $\{u, v\}$ with two new nodes u and v arrives, the master assigns both nodes to the same worker i^* with minimum load (lines 4-5) for pursuing Case M-1 and balancing loads. If an edge $\{u, v\}$ with one new node u (without loss of generality) arrives, the master assigns u to the worker $f(v)$, for Case M-1 to happen, as long as the load of $f(v)$ is not higher than $(1 + \theta)$ times of the load of worker i^*. Otherwise, load balancing is prioritized, and u is assigned to worker i^* with minimum load (lines 6-9). Once $f(u)$ and $f(v)$ are determined, each edge $\{u, v\}$ is sent to the worker(s) depending on $f(u)$ and $f(v)$ as in Algorithm 1, and the load of the corresponding worker(s) is updated (lines 10-16). Note that $f(u)$ and $f(v)$ are never changed once they are determined, Since the assignments by f are only in the master, along each edge to each worker, one bit indicating whether the edge is assigned to the worker or not should be sent.

Table 3: Case M-1 saves storage, communication, and computation costs, compared to Case M-2.

	Case M-1	Case M-2
storage (edge is stored in at most)	1 worker	2 workers
communication (edge is sent to)	1 worker	$\lvert\mathcal{W}\rvert$ workers
computation (COUNT() is called in)	1 worker	$\lvert\mathcal{W}\rvert$ workers

Advantages of DiSLR$_{\text{OPT}}$. By co-optimizing storage and load balancing, DiSLR$_{\text{OPT}}$ stores more uniques edges and thus produces more accurate estimates than DiSLR$_{\text{SIMPLE}}$, which is DiSLR using the simple modulo function as f. Although our explanation so far has focused on storage and load balancing, DiSLR$_{\text{OPT}}$ also improves upon DiSLR$_{\text{SIMPLE}}$ in terms of speed by increasing the chance of Case M-1, which saves not only storage but also communication and computation costs, as summarized in Table 3.

4.5 Lazy Aggregation

According to Algorithm 1, the updates of the estimates are sent to the aggregator either whenever a triangle is discovered (line 11) or whenever an edge is processed (line 13). Instead, as suggested in [18], the workers can aggregate the updates of the estimates locally. Then, they can send the aggregated updates to the aggregator periodically or when the corresponding estimates are queried if such less frequent updates are sufficient. By doing so, the communication cost between the workers and the aggregator are reduced.

4.6 Multiple Sources, Masters and Aggregators

Consider the case when edges are streamed from one or more sources to multiple masters without duplication. If every master uses the same non-adaptive node mapping function f [4] (e.g., the modulo function), we can run the masters independently in parallel without affecting the accuracy. This is because, in such a case, masters have no state and thus nothing to share with each other.

Multiple aggregators are required when outputs (i.e., 1 global triangle count and $|\mathcal{V}^{(t)}|$ local triangle counts) do not fit one machine or aggregation is a performance bottleneck. The computation and storage required for aggregation are distributed across multiple aggregators if all workers use the same hash function that maps each key (either '$*$' or a node id) to an aggregator to decide where to send each key-value pair (in lines 11 and 13 of Algorithm 1)

5 THEORETICAL ANALYSES

We theoretically analyze the accuracy of DiSLR in Section 5.1 and its time and space complexities in Section 5.2.

[4]A node mapping function f is non-adaptive if its mapping does not depend on any states. Algorithm 2 is adaptive since its mapping depends on the loads of workers.

5.1 Accuracy Analysis

We analyze the biases and variances of the estimates given by DiSLR. For the analyses, consider $\mathcal{G}^{(t)} = (\mathcal{V}^{(t)}, \mathcal{E}^{(t)})$, which is the graph consisting of the edges arriving at time t or earlier. We define $\bar{c}^{(t)}$ as $\bar{c}$ in the aggregator after edge $e^{(t)}$ is processed. Then, $\bar{c}^{(t)}$ is an estimate of $|\mathcal{T}^{(t)}|$, the global triangle count in $\mathcal{G}^{(t)}$. Likewise, for each node $u \in \mathcal{V}^{(t)}$, we define $c^{(t)}[u]$ as $c[u]$ in the aggregator after $e^{(t)}$ is processed. Then, each $c^{(t)}[u]$ is an estimate of $|\mathcal{T}^{(t)}[u]|$, the local triangle count of u in $\mathcal{G}^{(t)}$.

5.1.1 Bias Analysis. We first prove that DiSLR gives unbiased estimates, whose expected values are equal to the true triangle counts, as formalized in Theorem 5.1.

Theorem 5.1 (Unbiasedness of DiSLR). *At any time, the expected values of the estimates given by DiSLR are equal to the true global and local triangle counts. That is, in Algorithm 1,*

$$\boxed{\mathbb{E}[\bar{c}^{(t)}] = |\mathcal{T}^{(t)}|,} \ \forall t \geq 1$$

$$\boxed{\mathbb{E}[c^{(t)}[u]] = |\mathcal{T}^{(t)}[u]|,} \ \forall u \in \mathcal{V}^{(t)}, \ \forall t \geq 1.$$

Sketch of Proof. By Lemma 4.1, there is exactly one worker that can count each triangle. For each triangle, if the corresponding worker counts the triangle, the corresponding estimates are increased by the reciprocal of the probability that the triangle is counted (lines 11-13). This makes the expected increase in the estimates be exactly one for each triangle. Thus, the aggregated estimates have expected values equal to the true triangle counts. See Section A.1 of the supplementary document [1] for a full proof. ■

5.1.2 Variance Analysis. Having shown that $\bar{c}^{(t)}$ is an unbiased estimate of the global triangle count $|\mathcal{T}^{(t)}|$, we analyze its variance to give an intuition why the variance is smaller in DiSLR than in Tri-Fly [18]. The variance of each $c^{(t)}[u]$ can be analyzed in the same manner considering only the local triangles incident to node u. We first define the two types of triangle pairs illustrated in Figure 3.

Definition 5.2 (Type 1 Triangle Pair). A Type 1 triangle pair is two different triangles $\{u, v, w\}$ and $\{u, v, x\}$ sharing an edge $\{u, v\}$ satisfying $t_{wu} = \max(t_{uv}, t_{vw}, t_{wu})$ and $t_{xu} = \max(t_{uv}, t_{vx}, t_{xu})$.

Definition 5.3 (Type 2 Triangle Pair). A Type 2 triangle pair is two different triangles $\{u, v, w\}$ and $\{u, v, x\}$ sharing an edge $\{u, v\}$ satisfying $t_{vw} = \max(t_{uv}, t_{vw}, t_{wu})$ and $t_{xu} = \max(t_{uv}, t_{vx}, t_{xu})$.

The variance of estimate $\bar{c}^{(t)}$ depends on how the triangles in $\mathcal{T}^{(t)}$ are distributed across workers. By Lemma 4.1, there is exactly one worker that can count each triangle. Thus, for each worker $i \in \mathcal{W}$, let $\mathcal{T}_i^{(t)} \subset \mathcal{T}^{(t)}$ be the set of triangles that can be counted by i. Likewise, let $p_i^{(t)}$ and $q_i^{(t)}$ be the numbers of Type 1 pairs and Type 2 pairs, respectively, among the triangles in $\mathcal{T}_i^{(t)}$. Then, for each worker $i \in \mathcal{W}$, we define $z_i^{(t)}$ as

$$z_i^{(t)} := \max\left(0, |\mathcal{T}_i^{(t)}|\Big(\frac{(l_i^{(t)}-1)(l_i^{(t)}-2)}{k(k-1)} - 1\Big) + (p_i^{(t)} + q_i^{(t)})\frac{l_i^{(t)} - 1 - k}{k}\right),$$

where $l_i^{(t)}$ is the load l_i of each worker $i \in \mathcal{W}$ when $e^{(t)}$ arrives. This term is used to upper bound the variance of $\bar{c}^{(t)}$ in Theorem 5.4. According to the theorem, each worker's contribution

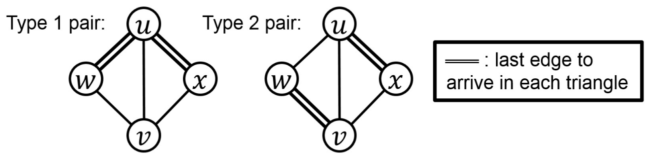


Figure 3: Type 1 and Type 2 triangle pairs.

to the variance decreases as the storage budget k increases, while the contribution increases as more edges, triangles, and Type 1 or 2 triangle pairs (whose discovering probabilities are positively correlated) are assigned to the worker, matching our intuition.

Theorem 5.4 (Variance of DiSLR). *At any time t, the variance of the estimate $\bar{c}^{(t)}$ of the global triangle count $|\mathcal{T}^{(t)}|$ in Algorithm 1 is upper bounded by the sum of $z_i^{(t)}$ in every worker $i \in \mathcal{W}$. That is,*

$$Var[\bar{c}^{(t)}] \leq \sum\nolimits_{i \in \mathcal{W}} z_i^{(t)}, \ \forall t \geq 1 \tag{2}$$

Sketch of Proof. Let $\bar{c}_i^{(t)}$ be the global triangle count sent from each worker i by time t. Then, by line 22, $\bar{c}^{(t)} = \sum_{i \in \mathcal{W}} \bar{c}_i^{(t)}$. Since $\bar{c}_i^{(t)}$ of each worker $i \in \mathcal{W}$ is independent from that of the other workers,

$$Var[\bar{c}^{(t)}] = \sum\nolimits_{i \in \mathcal{W}} Var[\bar{c}_i^{(t)}]. \tag{3}$$

Then, Theorem 5 in [9] is generalized for each $\bar{c}_i^{(t)}$ to $Var[\bar{c}_i^{(t)}] \leq z_i^{(t)}$. This generalization and Eq. (3) imply Eq. (2). ■

We compare the variance of $\bar{c}^{(t)}$ in DiSLR (i.e., Eq. (2)) to that of the estimate in Tri-Fly when a uniform random function is used as f. Lemma 5.5 states how rapidly each random variable in Eq. (2) decreases depending on the number of workers. In the lemma, $\mathbb{E}_f[q_i^{(t)}]$ can decrease faster than $O(q^{(t)}/|\mathcal{W}|)$ since more Type 2 triangle pairs are assigned to no worker[5] as more workers are used.

Lemma 5.5. *Assume $f : \mathcal{V} \to \mathcal{W}$ is a random function where $\mathbb{P}[f(u) = i] = 1/|\mathcal{W}|$ for each node $u \in \mathcal{V}$ and each worker $i \in \mathcal{W}$. Then, the following equations hold for any time $t \geq 1$:*

$$\mathbb{E}_f[|\mathcal{T}_i^{(t)}|] = O\left(\frac{|\mathcal{T}^{(t)}|}{|\mathcal{W}|}\right), \ \mathbb{E}_f[l_i^{(t)}] = O\left(\frac{t}{|\mathcal{W}|}\right),$$
$$\mathbb{E}_f[p_i^{(t)}] = O\left(\frac{p^{(t)}}{|\mathcal{W}|}\right), \ \mathbb{E}_f[q_i^{(t)}] = O\left(\frac{q^{(t)}}{|\mathcal{W}|^2}\right).$$

Proof. See Section A.2 of the supplementary document [1]. ■

If we assume that there is not much positive correlation between random variables, then for each $z_i^{(t)}$ in Eq. (2),

$$\mathbb{E}_f[z_i^{(t)}] \approx \mathbb{E}_f[|\mathcal{T}_i^{(t)}|]\Big(\frac{(\mathbb{E}_f[l_i^{(t)}]-1)(\mathbb{E}_f[l_i^{(t)}]-2)}{k(k-1)} - 1\Big)$$
$$+ (\mathbb{E}_f[p_i^{(t)}] + \mathbb{E}_f[q_i^{(t)}])\frac{\mathbb{E}_f[l_i^{(t)}] - 1 - k}{k} = O\left(\frac{|\mathcal{T}^{(t)}|t^2}{|\mathcal{W}|^3k^2} + \frac{p^{(t)}t}{|\mathcal{W}|^2k} + \frac{q^{(t)}t}{|\mathcal{W}|^3k}\right).$$

Then, by Theorem 5.4, the expected variance of $\bar{c}^{(t)}$ is

$$\mathbb{E}_f[Var[\bar{c}^{(t)}]] \approx \boxed{O\left(\frac{|\mathcal{T}^{(t)}|t^2}{|\mathcal{W}|^2k^2} + \frac{p^{(t)}t}{|\mathcal{W}|k} + \frac{q^{(t)}t}{|\mathcal{W}|^2k}\right).} \tag{4}$$

[5]A Type 2 triangle pair is assigned to no worker if the two triangles are assigned to different workers.

Table 4: Time and space complexities of processing first t edges in the input stream.

Time Complexity ($S = \min(t, k|\mathcal{W}|)$ and $L = \min(t|\mathcal{W}|, k|\mathcal{W}|)$)

Methods	Master	Workers (Total)	Aggregator
DiSLR (both)	$O(t\|\mathcal{W}\|)^*$	$O(tS)$	$O(\min(tS, \|\mathcal{T}^{(t)}\|))^*$
Tri-Fly [18]		$O(tL)$	$O(\min(tL, \|\mathcal{T}^{(t)}\| \cdot \|\mathcal{W}\|))^*$

Space Complexity

Methods	Master	Workers (Total)	Aggregator
DiSLR$_{\text{SIMPLE}}$	$O(\|\mathcal{W}\|)$	$O(\min(t, k\|\mathcal{W}\|))$	$O(\|\mathcal{V}^{(t)}\|)^*$
DiSLR$_{\text{OPT}}$	$O(\|\mathcal{V}^{(t)}\| + \|\mathcal{W}\|))$	$O(\min(t, k\|\mathcal{W}\|))$	
Tri-Fly [18]	$O(\|\mathcal{W}\|)$	$O(\min(t\|\mathcal{W}\|, k\|\mathcal{W}\|))$	

*can be distributed across multiple masters or aggregators (see Section 4.6)

On the other hand, the variance of the estimate in Tri-Fly (See Appendix A.2 of [18]) is

$$Var[\bar{c}^{(t)}] = O\left(\frac{|\mathcal{T}^{(t)}|t^2}{|\mathcal{W}|k^2} + \frac{p^{(t)}t}{|\mathcal{W}|k} + \frac{q^{(t)}t}{|\mathcal{W}|k}\right). \qquad (5)$$

Notice how rapidly the variances in DiSLR (Eq. (4)) and Tri-Fly (Eq. (5)) decrease depending on the number of workers (i.e., $|\mathcal{W}|$). In Eq. (4), only the second term is $O(1/|\mathcal{W}|)$ while the other terms are $O(1/|\mathcal{W}|^2)$. In Eq. (5), however, all the terms are $O(1/|\mathcal{W}|)$. This analysis gives an intuition why we can expect smaller variance of $\bar{c}^{(t)}$ in DiSLR than in Tri-Fly, especially when many workers are used. See Section 6.2 for empirical comparison of the variances.

In addition, we prove in Section B of the supplementary document [1] that the conditions for exact triangle counting in DiSLR is weaker than that in Tri-Fly, as confirmed empirically in Section 6.5.

5.2 Complexity Analyses

We summarize the time and space complexities of DiSLR$_{\text{SIMPLE}}$ (DiSLR with the modulo function as f), DiSLR$_{\text{OPT}}$, and Tri-Fly in Table 4. Detailed analyses with proofs are given in Section C of the supplementary document [1]. With a fixed storage budget k, the time complexity of DiSLR$_{\text{SIMPLE}}$ and DiSLR$_{\text{OPT}}$ is linear to the number of edges in the input stream, as confirmed empirically in Section 6.4. They also have a lower time complexity than Tri-Fly.

6 EXPERIMENTS

We review our experiments for answering the following questions:

- **Q1. Illustration of Theorems**: Does DiSLR give unbiased estimates? How do their variances scale with the number of workers?
- **Q2. Speed and Accuracy**: Is DiSLR faster and more accurate than its best competitors?
- **Q3. Scalability**: Does the running time of DiSLR is linear to the number of edges in the input graph stream?
- **Q4. Effects of Parameters**: How do the number of workers, storage budget, and parameter θ affect the accuracy of DiSLR?

In the supplementary document [1], we conduct experiments on fault tolerance (Section D) and evaluate the accuracy of DiSLR with more metrics (Section E) and competitors (Section F).

6.1 Experimental Settings

Machines: All experiments were conducted on a cluster of 40 machines with 3.47GHz Intel Xeon X5690 CPUs and 32GB RAM.
Graph Streams: We used the graphs listed in Table 5. We ignored self loops, parallel edges, and the directions of edges. We simulated

Table 5: Summary of the datasets used in our experiments.

Name	# Nodes	# Edges	Summary
ArXiv	34, 546	420, 877	Citation
Facebook	63, 731	817, 090	Friendship
Google	875, 713	4, 322, 051	Web
BerkStan	685, 230	6, 649, 470	Web
Youtube	3, 223, 589	9, 376, 594	Friendship
Flickr	2, 302, 925	22, 838, 276	Friendship
LiveJournal	3, 997, 962	34, 681, 189	Friendship
Orkut	3, 072, 441	117, 185, 083	Friendship
FriendSter	65, 608, 366	1, 806, 067, 135	Friendship
Random (**800GB**)	1, 000, 000	100, 000, 000, 000	Synthetic

graph streams by streaming the edges of the corresponding graph in a random order from the disk of the machine hosting the master.
Implementations: We implemented the following algorithms, which are the state-of-the-art algorithms that estimate **both global and local triangle counts**, commonly in C++ and MPICH 3.1:

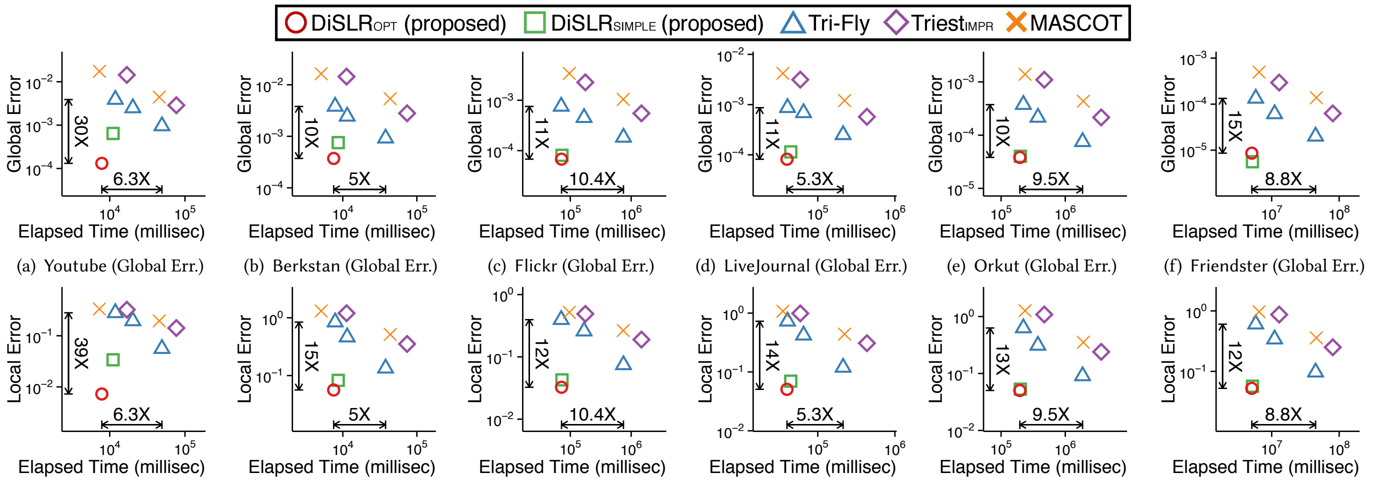
- **DiSLR$_{\text{SIMPLE}}$** and **DiSLR$_{\text{OPT}}$**: proposed distributed streaming algorithms using the modulo function and Algorithm 2, respectively, as the node mapping function f.
- **Tri-Fly** [18]: state-of-the-art distributed streaming algorithm.
- **Mascot** [13] and **Triest$_{\text{IMPR}}$** [9]: state-of-the-art single-machine streaming algorithms (see Section F of the supplementary document [1] for comparison with other single-machine algorithms)

For the distributed algorithms, we used one master and one aggregator hosted by the same machine. Workers are hosted by different machines (unless their number is greater than that of machines) using a part of the main memory of hosting machines as their local storage. In every algorithm, sampled edges were stored in the adjacency list format, and lazy aggregation, explained in Section 4.5, was used so that all estimates were aggregated once at the end of each input stream. We set θ in DiSLR$_{\text{OPT}}$ to 0.2, which gave the best accuracy in Section 6.5, unless otherwise stated.
Evaluation Metrics: We measured the accuracy of the considered algorithms at the end of each input stream. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be the graph at the end of the input stream. Then, for each node $u \in \mathcal{V}$, let $x[u]$ be the true local count of u in $\mathcal{G}$, and let $\hat{x}[u]$ be its estimate obtained by the evaluated algorithm. Likewise, let x and $\hat{x}$ be the true and estimated global triangle count, respectively. The accuracy of global triangle counting was evaluated using *global error*, defined as $\frac{|x-\hat{x}|}{1+x}$. For the accuracy of local triangle counting, we used *local error*, defined as $\frac{1}{|\mathcal{V}|}\sum_{u\in\mathcal{V}}\frac{|x[u]-\hat{x}[u]|}{1+x[u]}$, and *local RMSE*, defined as $\sqrt{\frac{1}{|\mathcal{V}|}\sum_{u\in\mathcal{V}}(x[u]-\hat{x}[u])^2}$. We also used Spearman's rank correlation coefficient [19] between $\{(u, x[u])\}_{u\in\mathcal{V}}$ and $\{(u, \hat{x}[u])\}_{u\in\mathcal{V}}$.

6.2 Q1. Illustration of Our Theorems

DiSLR gave unbiased estimates with small variances. Figure 1(c) in Section 1 illustrates Theorem 5.1, the unbiasedness of DiSLR. We obtained 10, 000 estimates of the global triangle count in the Google dataset using each distributed algorithm. We used 30 workers, and set k so that each worker stored up to 5% of the edges. As expected from Theorem 5.1, DiSLR$_{\text{OPT}}$ and DiSLR$_{\text{SIMPLE}}$ gave estimates whose averages were close to the true triangle count. The variance was the smallest in DiSLR$_{\text{OPT}}$, and the variance in DiSLR$_{\text{SIMPLE}}$ was smaller than that in Tri-Fly.

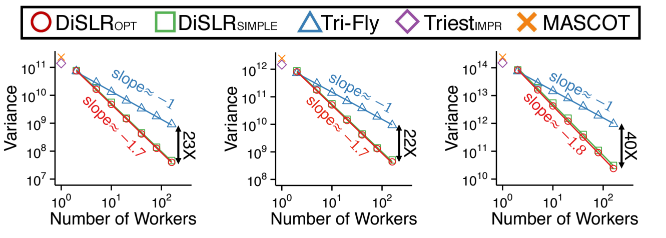
The variance in DiSLR dropped fast with the number of workers. Figure 5 illustrates Theorem 5.4, the variance of the



(a) Youtube (Global Err.) (b) Berkstan (Global Err.) (c) Flickr (Global Err.) (d) LiveJournal (Global Err.) (e) Orkut (Global Err.) (f) Friendster (Global Err.)

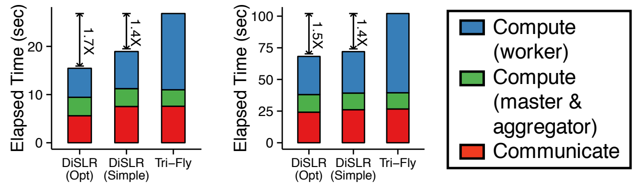
(g) Youtube (Local Err.) (h) Berkstan (Local Err.) (i) Flickr (Local Err.) (j) LiveJournal (Local Err.) (k) Orkut (Local Err.) (l) Friendster (Local Err.)

Figure 4: DiSLR significantly outperforms all competitors. DiSLR$_{\text{OPT}}$ (whose θ is fixed to 0.2) is up to 39× more accurate than its competitors with similar speeds. DiSLR$_{\text{OPT}}$ is up to 10.4× faster than its competitors while giving more accurate estimates.



(a) Arxiv Dataset (b) Facebook Dataset (c) Google Dataset

Figure 5: Variance of estimates decreases faster in DiSLR$_{\text{OPT}}$ and DiSLR$_{\text{SIMPLE}}$ than in Tri-Fly as we use more workers.



(a) Youtube Dataset (b) LiveJournal Dataset

Figure 6: DiSLR$_{\text{OPT}}$ reduces computation in workers and communication, which take most running time.

estimate of the global triangle count in DiSLR. As we scaled up the number of workers, the variance decreased faster in DiSLR$_{\text{OPT}}$ and DiSLR$_{\text{SIMPLE}}$ ($\approx |\mathcal{W}|^{-1.7}$) than in Tri-Fly ($\approx |\mathcal{W}|^{-1}$), as expected in Eq. (4) and Eq. (5) in Section 5.1.2. In each setting, k was set to 1,000, and the variance was estimated from 1,000 trials. With larger k or more machines, the slopes of DiSLR$_{\text{OPT}}$ and DiSLR$_{\text{SIMPLE}}$ became steeper leading to zero variance, as in Figure 7.

6.3 Q2. Speed and Accuracy

We measured the speed and accuracy of the considered algorithms with different storage budgets.[6] We used 30 workers for each distributed streaming algorithm. To compare their speeds independently of the speed of the input stream, we measured the time taken by each algorithm to process edges, ignoring the time taken to wait for the arrival of edges in the input stream. In Figure 4, we report the evaluation metrics and elapsed times averaged over 10 trials in the Friendster dataset and over 100 trials in other large datasets.

DiSLR gave the best trade-off between speed and accuracy. DiSLR$_{\text{OPT}}$ was up to **10.4× faster** than its competitors while giving more accurate estimates. Moreover, DiSLR$_{\text{OPT}}$ was up to **30× and 39× more accurate** than its competitors with similar speeds in terms of global error and local error, respectively. We obtained similar results in terms of local RMSE and rank correlation (see Section E of the supplementary document [1]). Between our algorithms, DiSLR$_{\text{OPT}}$ was up to 1.4× faster and 4.9× more accurate than DiSLR$_{\text{SIMPLE}}$. Figure 6 shows how much time was taken for (1) computation in the master and aggregator, (2) computation in the slowest worker, and (3) communication between machines, respectively, in each algorithm. DiSLR$_{\text{OPT}}$ decreased both computation and communication costs compared to DiSLR$_{\text{SIMPLE}}$ and Tri-Fly.

[6] $k = 5\%$ of the number of edges in each dataset in DiSLR$_{\text{SIMPLE}}$ and DiSLR$_{\text{OPT}}$. $k = \{2\%, 5\%, 20\%\}$ in Tri-Fly. $k = \{5\%, 40\%\}$ in Triest$_{\text{IMPR}}$ and Mascot.

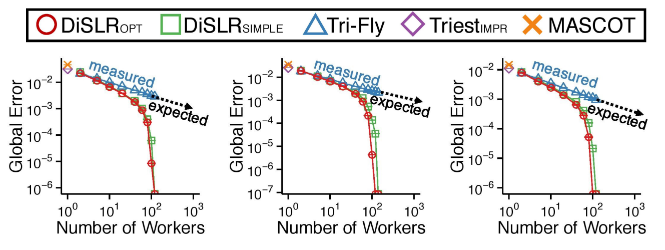
6.4 Q3. Scalability

We measured how the running times of DiSLR$_{\text{OPT}}$ and DiSLR$_{\text{SIMPLE}}$ scale with the number of edges in the input stream. We used 30 workers with fixed storage budget $k = 10^7$, and we measured their running times independently of the speed of the input stream, as in Section 6.3.

DiSLR scaled linearly and handled terabyte-scale graphs. Figure 1(b) in Section 1 shows the results with Erdős-Rényi random graph streams with 1 million nodes and different numbers of edges. Note that the largest stream has 100 billion edges, which are 800GB. DiSLR$_{\text{OPT}}$ and DiSLR$_{\text{SIMPLE}}$ scaled linearly with the size of the input stream, as we expect in Section 5.2. We obtained the same linear scalability using graph streams with realistic structure in Section G of the supplementary document [1].

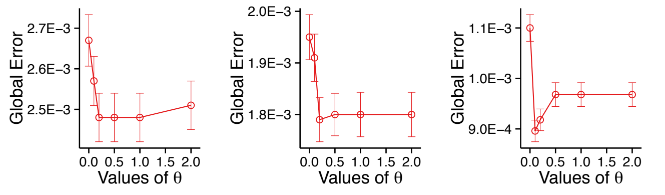
6.5 Q4. Effects of Parameters on Accuracy

We explored the effects of the parameters on the accuracies of the considered algorithms. As a default setting, we used 30 workers for the distributed streaming algorithms and set k to 2% of the number of edges for each dataset and θ to 0.2. When the effect of a parameter was analyzed, the others were fixed to their default values. We reported results with global error as the evaluation



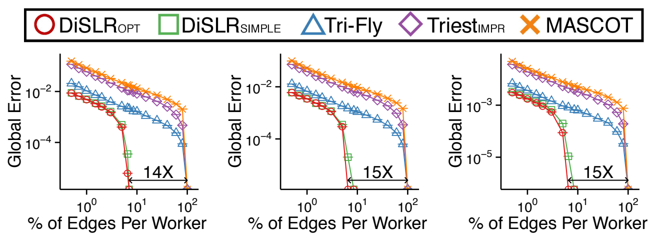
(a) Arxiv Dataset (b) Facebook Dataset (c) Google Dataset

Figure 7: Estimation error decreases faster in DiSLR than in Tri-Fly as we use more workers. The error eventually becomes zero in DiSLR, while it does not in Tri-Fly.



(a) Arxiv Dataset (b) Facebook Dataset (c) Google Dataset

Figure 8: Estimation error in DiSLR$_{\text{OPT}}$ is smallest when θ is around 0.2, while the error is not very sensitive to θ.



(a) Arxiv Dataset (b) Facebook Dataset (c) Google Dataset

Figure 9: Estimation error decreases faster in DiSLR than in the competitors as we increase storage budget k. For exact estimation, DiSLR requires 14× smaller k than the others.

metric but obtained consistent results with the other metrics. We measured it 1,000 times in each setting and reported the average. In Figures 7-9, the error bars denote standard errors.

As the number of workers increased, the estimation error dropped faster in DiSLR than in the baselines. As seen in Figure 7, the estimation errors of DiSLR$_{\text{OPT}}$ and DiSLR$_{\text{SIMPLE}}$ became 0 with about 100 workers. However, that of Tri-Fly dropped slowly with expectation that it never becomes zero with a finite number of workers (see Appendix A.2 of [18]).

As storage budget increased, the estimation error dropped faster in DiSLR than in the baselines. As seen in Figure 9, the estimation errors of DiSLR$_{\text{OPT}}$ and DiSLR$_{\text{SIMPLE}}$ became 0 when each worker could store about 7% of the edges in each dataset. However, the estimation errors of the baselines became zero only when each worker could store 100% of the edges in each dataset.

DiSLR$_{\text{OPT}}$ was most accurate when θ was around 0.2, as seen in Figure 8. The estimation error, however, was not very sensitive to the value of θ if θ was at least 0.2.

7 CONCLUSION

We propose DiSLR, a distributed streaming algorithm for estimating the counts of global and local triangles. By minimizing the redundant use of distributed computational and storage resources (Properties P1-P3), DiSLR offers the following advantages:

- **Accurate**: DiSLR is up to *39× more accurate* than its similarly fast competitors (Figure 4). DiSLR gives exact estimates within *14× smaller storage budgets* than its competitors (Figure 9).
- **Fast**: DiSLR is up to *10.4× faster* than its competitors while giving more accurate estimates (Figure 4). DiSLR scales linearly with the size of the input stream (Figure 1(b)), reducing both computation and communication costs (Figure 6).
- **Theoretically Sound**: DiSLR gives unbiased estimates (Theorem 5.1). Their variances drop faster than its competitors' as the number of machines is scaled up (Theorem 5.4 and Figure 5).

REFERENCES

[1] 2018. Supplementary Document. Available online: http://goo.gl/LrwHvB. (2018).
[2] Nesreen K Ahmed, Nick Duffield, Jennifer Neville, and Ramana Kompella. 2014. Graph sample and hold: A framework for big-graph analytics. In *KDD*.
[3] Nesreen K. Ahmed, Nick Duffield, Theodore L. Willke, and Ryan A. Rossi. 2017. On Sampling from Massive Graph Streams. *PVLDB* 10, 11 (2017), 1430–1441.
[4] Shaikh Arifuzzaman, Maleq Khan, and Madhav Marathe. 2013. PATRIC: A parallel algorithm for counting triangles in massive networks. In *CIKM*.
[5] Ziv Bar-Yossef, Ravi Kumar, and D Sivakumar. 2002. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *SODA*.
[6] Luca Becchetti, Paolo Boldi, Carlos Castillo, and Aristides Gionis. 2008. Efficient semi-strefaming algorithms for local triangle counting in massive graphs. In *KDD*.
[7] Jonathan Cohen. 2008. Trusses: Cohesive subgraphs for social network analysis. *National Security Agency Technical Report* 16 (2008).
[8] Jonathan Cohen. 2009. Graph twiddling in a mapreduce world. *Computing in Science & Engineering* 11, 4 (2009), 29–41.
[9] Lorenzo De Stefani, Alessandro Epasto, Matteo Riondato, and Eli Upfal. 2016. TRIEST: Counting Local and Global Triangles in Fully-dynamic Streams with Fixed Memory Size. In *KDD*.
[10] Alessandro Epasto, Silvio Lattanzi, Vahab Mirrokni, Ismail Oner Sebe, Ahmed Taei, and Sunita Verma. 2015. Ego-net community mining applied to friend suggestion. *PVLDB* 9, 4 (2015), 324–335.
[11] Madhav Jha, Comandur Seshadhri, and Ali Pinar. 2013. A space efficient streaming algorithm for triangle counting using the birthday paradox. In *KDD*.
[12] Konstantin Kutzkov and Rasmus Pagh. 2013. On the streaming complexity of computing local clustering coefficients. In *WSDM*.
[13] Yongsub Lim and U Kang. 2015. Mascot: Memory-efficient and accurate sampling for counting local triangles in graph streams. In *KDD*.
[14] Mark EJ Newman. 2003. The structure and function of complex networks. *SIAM review* 45, 2 (2003), 167–256.
[15] Ha-Myung Park, Sung-Hyon Myaeng, and U Kang. 2016. Pte: Enumerating trillion triangles on distributed systems. In *KDD*.
[16] Aduri Pavan, Kanat Tangwongan, and Srikanta Tirthapura. 2013. Parallel and distributed triangle counting on graph streams. *Technical report, IBM* (2013).
[17] Aduri Pavan, Kanat Tangwongsan, Srikanta Tirthapura, and Kun-Lung Wu. 2013. Counting and sampling triangles from a graph stream. *PVLDB* 6, 14 (2013), 1870–1881.
[18] Kijung Shin, Mohammad Hammoud, Euiwoong Lee, Jinoh Oh, and Christos Faloutsos. 2018. Tri-Fly: Distributed Estimation of Global and Local Triangle Counts in Graph Streams. In *PAKDD*.
[19] Charles Spearman. 1904. The proof and measurement of association between two things. *The American journal of psychology* 15, 1 (1904), 72–101.
[20] Siddharth Suri and Sergei Vassilvitskii. 2011. Counting triangles and the curse of the last reducer. In *WWW*.
[21] Kanat Tangwongsan, Aduri Pavan, and Srikanta Tirthapura. 2013. Parallel triangle counting in massive streaming graphs. In *CIKM*.
[22] Charalampos E Tsourakakis. 2008. Fast counting of triangles in large real networks without counting: Algorithms and laws. In *ICDM*.
[23] Charalampos E Tsourakakis, Petros Drineas, Eirinaios Michelakis, Ioannis Koutis, and Christos Faloutsos. 2011. Spectral counting of triangles via element-wise sparsification and triangle-based link recommendation. *Social Network Analysis and Mining* 1, 2 (2011), 75–81.
[24] Charalampos E Tsourakakis, U Kang, Gary L Miller, and Christos Faloutsos. 2009. Doulion: counting triangles in massive graphs with a coin. In *KDD*.
[25] Nan Wang, Jingbo Zhang, Kian-Lee Tan, and Anthony KH Tung. 2010. On triangulation-based dense neighborhood graph discovery. *PVLDB* 4, 2 (2010), 58–68.
[26] Stanley Wasserman and Katherine Faust. 1994. *Social network analysis: Methods and applications.* Vol. 8. Cambridge university press.
[27] Duncan J Watts and Steven H Strogatz. 1998. Collective dynamics of fismall-worldfinetworks. *nature* 393, 6684 (1998), 440–442.