Think before You Discard: Accurate Triangle Counting in Graph Streams with Deletions

Kijung Shin[1(✉)], Jisu Kim[2], Bryan Hooi[2], and Christos Faloutsos[1]

School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA
[1]{kijungs,christos}@cs.cmu.edu, [2]{jisuk1,bhooi}@andrew.cmu.edu

Abstract. Given a stream of edge additions and deletions, how can we estimate the count of triangles in it? If we can store only a subset of the edges, how can we obtain unbiased estimates with small variances?
Counting triangles (i.e., cliques of size three) in a graph is a classical problem with applications in a wide range of research areas, including social network analysis, data mining, and databases. Recently, streaming algorithms for triangle counting have been extensively studied since they can naturally be used for large dynamic graphs. However, existing algorithms cannot handle edge deletions or suffer from low accuracy.
Can we handle edge deletions while achieving high accuracy? We propose THINKD, which accurately estimates the counts of global triangles (i.e., all triangles) and local triangles associated with each node in a fully dynamic graph stream with edge additions and deletions. Compared to its best competitors, THINKD is **(a) Accurate:** up to *4.3× more accurate* within the same memory budget, **(b) Fast:** up to *2.2× faster* for the same accuracy requirements, and **(c) Theoretically sound:** always maintaining *unbiased* estimates with small variances.

Keywords: Triangle Counting, Local Triangles, Streaming Algorithms, Fully Dynamic Graph Streams, Edge Deletions

1 Introduction

Given a fully dynamic graph stream with edge additions and deletions, how can we accurately estimate the count of triangles in it with fixed memory size?

The count of triangles (i.e., cliques of size three) is a key primitive in graph analysis with a wide range of applications, including spam/anomaly detection [5,14], link recommendation [8,22], community detection [6], degeneracy estimation [18], and query optimization [3]. In particular, many important metrics in social network analysis, including the clustering coefficient [24], the transitivity ratio [15], and the triangle connectivity [4], are based on the count of triangles.

Many real graphs are best represented as a sequence of edge additions and deletions, and they often need to be processed in real time. For example, many social networking service companies aim to detect fraud or spam as quickly as possible in their online social networks, which evolve indefinitely with both edge additions and deletions. Another example is to examine graphs of data traffic and improve the network performance in real time.

Table 1: Comparison of streaming algorithms for triangle counting. Notice that **ThinkD is accurate while satisfying all the criteria.**

	TRIEST$_{FD}$ [7]	ESD [10]	Other Local [14]	Other Local [7,17]*	Other Global [2,11,16,20]	THINKD$_{FAST}$ (Proposed)	THINKD$_{ACC}$ (Proposed)
Local Triangles	∨		∨	∨		∨	∨
Large Graphs**	∨		∨	∨	∨	∨	∨
Edge Deletions	∨	∨				∨	∨
Accuracy***	-	-	∨	⩔	?	∨	⩔

*TRIEST$_{IMPR}$ [7], **graphs that do not fit in memory, *** ⩔: highest, ∨: high, ?: highest-low. -: low

As a result, there has been great interest in graph stream algorithms, which gradually update their outputs as each edge insertion or deletion is received rather than operating on the entire graph at once. However, existing streaming algorithms for triangle counting focus on insertion-only streams [2,11,14,16,17,19] or greatly sacrifice accuracy to support edge deletions [7,10,13].

In this work, we propose THINKD (**Think** before you **D**iscard), an accurate streaming algorithm for triangle counting in a fully dynamic graph stream with both edge additions and deletions. THINKD maintains and updates estimates of the counts of global triangles (i.e., all triangles) and local triangles incident to each node. THINKD is named after the fact that, upon receiving each edge addition or deletion, THINKD uses it to improve its estimates even if the edge is about to be discarded without being stored. This allows THINKD to achieve higher accuracy than if it were to only use edges in memory for estimation. As a result, our proposed algorithm THINKD has the following strengths:

- **Accurate**: THINKD gives up to *4*× and *4.3*× *smaller estimation errors* for global and local triangle counts, respectively, than its best competitors within the same memory budget (Fig. 2).
- **Fast**: THINKD *scales linearly* with the size of the input stream (Fig. 1, Corollary 1, and Theorem 4). Especially, THINKD is up to *2.2*× *faster* than its best competitors with similar accuracies (Fig. 3).
- **Theoretically Sound**: We prove the formulas for the bias and variance of the estimates provided by THINKD (Theorems 1 and 2). In particular, we show that THINKD always maintains *unbiased* estimates (Fig. 1).

Reproducibility: The source code and datasets used in the paper are available at `http://www.cs.cmu.edu/~kijungs/codes/thinkd/`.

In Sect. 2, we review related work. In Sect. 3, we present notations and the problem definition. In Sect. 4, we describe our proposed algorithm THINKD. After providing experimental results in Sect. 5, we conclude in Sect. 6.

2 Related Work

See Table 1 for a comparison of streaming algorithms for triangle counting. Streaming algorithms for triangle counting in insertion-only graph streams have

Table 2: Table of frequently-used symbols.

	Symbol	Definition
Notations for Fully Dynamic Graph Streams (Sect. 3)	$e^{(t)} = (\{u, v\}, \delta)$	change in the input graph $\mathcal{G}$ at time t
	$\mathcal{G}^{(t)} = (\mathcal{V}^{(t)}, \mathcal{E}^{(t)})$	graph $\mathcal{G}$ at time t
	$\{u, v\}$	edge between nodes u and v
	$\{u, v, w\}$	triangle with nodes u, v, and w
	$\mathcal{T}^{(t)}$	set of global triangles in $\mathcal{G}^{(t)}$
	$\mathcal{T}^{(t)}[u]$	set of local triangles of node u in $\mathcal{G}^{(t)}$
Notations for Algorithms and Analyses (Sect. 4)	$\mathcal{S}$	set of sampled edges
	$\hat{\mathcal{N}}[u]$	set of neighbors of node u in $\mathcal{S}$
	$\bar{c}$	estimate of the count of global triangles
	$c[u]$	estimate of the count of local triangles of node u
	r	sampling probability in THINKD$_{\text{FAST}}$
	k	maximum number of sampled edges in THINKD$_{\text{ACC}}$
	$\mathcal{A}^{(t)}$	set of added triangles at time t
	$\mathcal{D}^{(t)}$	set of deleted triangles at time t

been studied extensively, including multi-pass [12,21] or single-pass [2,11,16,20] algorithms for the count of global triangles, and multi-pass [5] or single-pass [7,14,17,19] algorithms for the counts of both global and local triangles.

The first algorithm for triangle counting in fully dynamic graph streams with edge deletions was proposed in [13]. The algorithm estimates the count of global triangles by making a single pass over the input stream. However, the algorithm is inapplicable to real-time applications since it expensively computes an estimate once at the end of the stream instead of always maintaining an estimate. Although ESD [10] maintains and updates an estimate of the global triangle count, its scalability is limited since it requires the entire input graph to be stored in memory. TRIEST$_{\text{FD}}$ [7], which maintains and updates estimates of both global and local triangle counts, scales better than ESD since TRIEST$_{\text{FD}}$ samples edges within a given memory budget and discards the other edges. However, TRIEST$_{\text{FD}}$, which simply discards those unsampled edges, is significantly less accurate than our proposed algorithm THINKD, which utilizes those unsampled edges to update estimates before discarding them. Although the idea of using unsampled edges has been considered for insertion-only streams [7,14,17,19], applying the idea to fully dynamic graph streams has remained unexplored.

3 Notations and Problem Definition

Notations: Table 2 lists the symbols frequently used in the paper. Consider an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with nodes $\mathcal{V}$ and edges $\mathcal{E}$. Each edge $\{u, v\} \in \mathcal{E}$ connects two distinct nodes $u \neq v \in \mathcal{V}$. We say a subset $\{u, v, w\} \subset \mathcal{V}$ of size 3 is a *triangle* if every pair of distinct nodes u, v, and w is connected by an edge in $\mathcal{E}$. We denote the set of *global triangles* (i.e., all triangles) in $\mathcal{G}$ by $\mathcal{T}$ and the set of *local triangles* of each node $u \in \mathcal{V}$ (i.e., all triangles containing u) by $\mathcal{T}[u] \subset \mathcal{T}$.

Assume the graph $\mathcal{G}$ evolves from the empty graph. We consider the *fully dynamic graph stream* representing the sequence of changes in $\mathcal{G}$, and denote the stream by $(e^{(1)}, e^{(2)}, ...)$. For each $t \in \{1, 2, ...\}$, the pair $e^{(t)} = (\{u, v\}, \delta)$ of an edge $\{u, v\}$ and a sign $\delta \in \{+, -\}$ denotes the change in $\mathcal{G}$ at time t. Specifically, $(\{u, v\}, +)$ indicates the addition of a new edge $\{u, v\} \notin \mathcal{E}$, and $(\{u, v\}, -)$ indicates the deletion of an existing edge $\{u, v\} \in \mathcal{E}$. We use $\mathcal{G}^{(t)} = (\mathcal{V}^{(t)}, \mathcal{E}^{(t)})$ to indicate $\mathcal{G}$ at time t. That is,

$$\mathcal{E}^{(0)} = \emptyset \qquad \text{and} \qquad \mathcal{E}^{(t)} = \begin{cases} \mathcal{E}^{(t-1)} \cup \{\{u, v\}\}, & \text{if } e^{(t)} = (\{u, v\}, +), \\ \mathcal{E}^{(t-1)} \setminus \{\{u, v\}\}, & \text{if } e^{(t)} = (\{u, v\}, -). \end{cases}$$

Lastly, we let $\mathcal{T}^{(t)}$ denote the set of global triangles in $\mathcal{G}^{(t)}$ and $\mathcal{T}^{(t)}[u] \subset \mathcal{T}^{(t)}$ denote the set of local triangles of each node $u \in \mathcal{V}^{(t)}$ in $\mathcal{G}^{(t)}$.

Problem Definition (Problem 1): In this work, we address the problem of estimating the counts of global and local triangles in a fully dynamic graph stream. We assume the standard data stream model where the elements in the input stream, which may not fit in memory, can be accessed once in the given order unless they are explicitly stored in memory.

Problem 1 (Global and Local Triangle Counting in a Fully Dynamic Graph Stream).

- **Given:** a fully dynamic graph stream $(e^{(1)}, e^{(2)}, ...)$
  (i.e., sequence of edge additions and deletions in graph $\mathcal{G}$)
- **Maintain:** estimates of global triangle count $|\mathcal{T}^{(t)}|$ and local triangle counts $\{(u, |\mathcal{T}^{(t)}[u]|)\}_{u \in \mathcal{V}^{(t)}}$ of graph $\mathcal{G}^{(t)}$ for current $t \in \{1, 2, ...\}$
- **to Minimize:** the estimation errors.

We follow a general approach of reducing the biases and variances of estimates simultaneously rather than minimizing a specific measure of estimation error.

4 Proposed Method: Think before You Discard (ThinkD)

We propose ThinkD (**Think** before you **D**iscard), which estimates the counts of global and local triangles in a fully dynamic graph stream. For estimation with limited memory, ThinkD samples edges and maintains those sampled edges, while discarding the other edges. The main idea of ThinkD is to fully utilize unsampled edges before they are discarded. Specifically, whenever each element in the input stream arrives, ThinkD first updates its estimates using the element. After that, if the element is an addition of an edge, ThinkD decides whether to sample the edge or not.

We present two versions of ThinkD and theoretically analyze their accuracies and complexities. To this end, we use $\bar{c}$ to denote the maintained estimate of the count of global triangles. Likewise, for each node u, we use $c[u]$ to denote the maintained estimate of the count of local triangles of node u. In addition, we let $\mathcal{S}$ be the set of currently sampled edges, and for each node u, we let $\hat{\mathcal{N}}[u]$ be the set of neighbors of u in the graph composed of the edges in $\mathcal{S}$.

Algorithm 1: THINKD$_{\text{FAST}}$: Simple and Fast Version of THINKD

Inputs : fully dynamic graph stream: $(e^{(1)}, e^{(2)}, ...)$, sampling probability: r
Outputs: estimate of the global triangle count: $\bar{c}$
estimates of the local triangle counts: $c[u]$ for each node u

1 $\mathcal{S} \leftarrow \emptyset$
2 **for each** element $e^{(t)} = (\{u, v\}, \delta)$ in the input stream **do**
3 UPDATE$(\{u, v\}, \delta)$
4 **if** $\delta = +$ **then** INSERT$(\{u, v\})$
5 **else if** $\delta = -$ **then** DELETE$(\{u, v\})$
6 **Procedure** UPDATE$(\{u, v\}, \delta)$:
7 **for each** common neighbor $w \in \hat{\mathcal{N}}[u] \cap \hat{\mathcal{N}}[v]$ **do**
8 **if** $\delta = +$ **then** increase $\bar{c}$, $c[u]$, $c[v]$, and $c[w]$ by $1/r^2$
9 **else if** $\delta = -$ **then** decrease $\bar{c}$, $c[u]$, $c[v]$, and $c[w]$ by $1/r^2$
10 **Procedure** INSERT$(\{u, v\})$:
11 **if** a random number in Bernoulli(r) is 1 **then** $\mathcal{S} \leftarrow \mathcal{S} \cup \{\{u, v\}\}$
12 **Procedure** DELETE$(\{u, v\})$:
13 **if** $\{u, v\} \in \mathcal{S}$ **then** $\mathcal{S} \leftarrow \mathcal{S} \setminus \{\{u, v\}\}$

4.1 ThinkD$_{\text{fast}}$: Simple and Fast Version of ThinkD

THINKD$_{\text{FAST}}$, which is a simple and fast version of THINKD, is described in Algorithm 1. THINKD$_{\text{FAST}}$ initially has no sampled edges (line 1). Whenever each element $(\{u, v\}, \delta)$ of the input stream arrives (line 2), THINKD$_{\text{FAST}}$ first updates its estimates by calling the procedure UPDATE (line 3). Then, if the element is an addition (i.e., $\delta = +$), THINKD$_{\text{FAST}}$ samples the edge $\{u, v\}$ with a given sampling probability r (line 11) by calling the procedure INSERT (line 4). If the element is a deletion (i.e., $\delta = -$), THINKD$_{\text{FAST}}$ removes the edge $\{u, v\}$ from the existing samples (line 13) by calling the procedure DELETE (line 5).

In the procedure UPDATE, THINKD$_{\text{FAST}}$ finds the triangles connected by the arrived edge $\{u, v\}$ and two edges from the existing samples $\mathcal{S}$ (line 7). To this end, THINKD uses the fact that each common neighbor w of the nodes u and v in the graph composed of the sampled edges in $\mathcal{S}$ indicates the existence of such a triangle $\{u, v, w\}$. In the case of additions (i.e., $\delta = +$), since such triangles are new triangles added to the input stream, THINKD$_{\text{FAST}}$ increases the estimates of the global count and the corresponding local counts (line 8). In the case of deletions (i.e., $\delta = -$), since such triangles are those removed from the input stream, THINKD$_{\text{FAST}}$ decreases the estimates of the global count and the corresponding local counts (line 9). Notice that the amount of change per triangle is $1/r^2$, which is the reciprocal of the probability that each added or deleted triangle is discovered by THINKD$_{\text{FAST}}$. Note that each such triangle $\{u, v, w\}$ is discovered if and only if $\{w, u\}$ and $\{v, w\}$ are in $\mathcal{S}$, whose probability is r^2, as formalized in Lemma 1. This makes the expected amount of changes in the corresponding estimates for each such triangle be exactly one and thus makes THINKD$_{\text{FAST}}$ give unbiased estimates, as explained in detail in Sect. 4.3.

Lemma 1 (Discovery Probability of Triangles in ThinkD$_{\text{fast}}$). *In* THINKD$_{\text{FAST}}$*, any two distinct edges in graph* $\mathcal{G}^{(t)} = (\mathcal{V}^{(t)}, \mathcal{E}^{(t)})$ *are sampled with probability* r^2*. That is, if we let* $\mathcal{S}^{(t)}$ *be* $\mathcal{S}$ *in Algorithm 1 after the* t*-th element* $e^{(t)}$ *is processed, then*

$$Pr[\{u,v\} \in \mathcal{S}^{(t)} \cap \{w,x\} \in \mathcal{S}^{(t)}] = r^2, \ \forall t \geq 1, \ \forall \{u,v\} \neq \{w,x\} \in \mathcal{E}^{(t)}. \quad (1)$$

Proof. Eq (1) holds since each edge is sampled independently with probability r. See Sect. A.1 of the supplementary document [1] for a formal proof. ■

(Dis)advantages of ThinkD$_{\text{fast}}$: Due to its simplicity, THINKD$_{\text{FAST}}$ is faster than its competitors, as shown empirically in Sect. 5.4. However, it is less accurate than THINKD$_{\text{ACC}}$, described in the following subsection, since it may discard edges even when memory is not full, leading to avoidable loss of information.

4.2 ThinkD$_{\text{acc}}$: Accurate Version of ThinkD

THINKD$_{\text{ACC}}$, which is an accurate version of THINKD, is described in Algorithm 2. Unlike THINKD$_{\text{FAST}}$, which may discard edges even when memory is not full, THINKD$_{\text{ACC}}$ maintains as many samples as possible within a given memory budget k (≥ 2) to minimize information loss.

To this end, THINKD$_{\text{ACC}}$ uses a sampling method called Random Pairing (RP) [9]. Given a fully dynamic stream with deletions, and a memory budget k, RP maintains at most k samples while satisfying the uniformity of the samples. That is, if we let $\mathcal{E}$ be the set of edges remaining (without being deleted) in the input stream so far and $\mathcal{S} \subset \mathcal{E}$ be the set of samples being maintained by RP, then the following equations hold:

$$|\mathcal{S}| \leq k \qquad \text{and} \qquad Pr[\mathcal{S} = \mathcal{A}] = Pr[\mathcal{S} = \mathcal{B}], \ \forall \mathcal{A} \neq \mathcal{B} \subset \mathcal{E} \text{ s.t. } |\mathcal{A}| = |\mathcal{B}|.$$

Updating the set $\mathcal{S}$ of samples using RP is described in lines 10-23. Whenever a deletion of an edge arrives, RP increases n_b or n_g depending on whether the edge is in $\mathcal{S}$ or not (lines 22 and 23). Roughly speaking, n_b and n_g denote the number of deletions that need to be "compensated" by additions (lines 16-18). If there is no deletion to compensate, RP processes each addition of an edge as in Reservoir Sampling [23]. That is, if memory is not full (i.e., $|\mathcal{S}| < k$), RP adds the new edge to $\mathcal{S}$ (line 13), while otherwise, RP replaces a random edge in $\mathcal{S}$ with the new edge with a certain probability (lines 14-15). We refer to [9] for the intuition behind the compensation and the details of RP; and we focus on how to use RP for triangle counting in the rest of this section.

Updating the estimates in THINKD$_{\text{ACC}}$ is the same as that in THINKD$_{\text{FAST}}$ except for the amount of change per triangle (lines 8 and 9), which is the reciprocal of the probability that each added or deleted triangle is discovered. When each element $e^{(t)} = (\{u,v\}, \delta)$ arrives, each added or deleted triangle $\{u,v,w\}$ is discovered if and only if $\{w,u\}$ and $\{v,w\}$ are in $\mathcal{S}$. As shown in Lemma 2, if we let $y = \min(k, |\mathcal{E}| + n_b + n_g)$, then the probability of such an event is

$$p(|\mathcal{E}|, n_b, n_g) := \frac{y}{|\mathcal{E}| + n_b + n_g} \times \frac{y-1}{|\mathcal{E}| + n_b + n_g - 1}. \quad (2)$$

Algorithm 2: THINKD$_{\text{ACC}}$: Accurate Version of THINKD

Inputs : fully dynamic graph stream: $(e^{(1)}, e^{(2)}, ...)$, memory budget: k (≥ 2)
Outputs: estimate of the global triangle count: $\bar{c}$
estimates of the local triangle counts: $c[u]$ for each node u

$\mathcal{S} \leftarrow \emptyset$, $|\mathcal{E}| \leftarrow 0$, $n_b \leftarrow 0$, $n_g \leftarrow 0$
for each element $e^{(t)} = (\{u, v\}, \delta)$ in the input stream **do**
  UPDATE($\{u, v\}, \delta$)
  if $\delta = +$ **then** INSERT($\{u, v\}$)
  else if $\delta = -$ **then** DELETE($\{u, v\}$)
Procedure UPDATE($\{u, v\}, \delta$):
  for each common neighbor $w \in \hat{\mathcal{N}}[u] \cap \hat{\mathcal{N}}[v]$ **do**
    if $\delta = +$ **then** increase $\bar{c}$, $c[u]$, $c[v]$, and $c[w]$ by $1/p(|\mathcal{E}|, n_b, n_g)$
    else if $\delta = -$ **then** decrease $\bar{c}$, $c[u]$, $c[v]$, and $c[w]$ by $1/p(|\mathcal{E}|, n_b, n_g)$
Procedure INSERT($\{u, v\}$):
  $|\mathcal{E}| \leftarrow |\mathcal{E}| + 1$
  if $n_b + n_g = 0$ **then**
    if $|\mathcal{S}| < k$ **then** $\mathcal{S} \leftarrow \mathcal{S} \cup \{\{u, v\}\}$
    else if *a random number in* Bernoulli$(k/|\mathcal{E}|)$ is 1 **then**
      replace a random edge in $\mathcal{S}$ with $\{u, v\}$
  else if *a random number in* Bernoulli$(n_b/(n_b + n_g))$ is 1 **then**
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{\{u, v\}\}$, $n_b \leftarrow n_b - 1$
  else $n_g \leftarrow n_g - 1$
Procedure DELETE($\{u, v\}$):
  $|\mathcal{E}| \leftarrow |\mathcal{E}| - 1$
  if $\{u, v\} \in \mathcal{S}$ **then**
    $\mathcal{S} \leftarrow \mathcal{S} \setminus \{\{u, v\}\}$, $n_b \leftarrow n_b + 1$
  else $n_g \leftarrow n_g + 1$

Lemma 2 (Discovery Probability of Triangles in ThinkD$_{\text{acc}}$). *In* THINKD$_{\text{ACC}}$*, any two distinct edges in graph* $\mathcal{G}^{(t)} = (\mathcal{V}^{(t)}, \mathcal{E}^{(t)})$ *are sampled with probability as in Eq.* (2)*. That is, if we let* $p^{(t)}$ *and* $\mathcal{S}^{(t)}$ *be the values of Eq.* (2) *and* $\mathcal{S}$*, resp., in Algorithm 2 after the* t*-th element* $e^{(t)}$ *is processed, then*

$$Pr[\{u, v\} \in \mathcal{S}^{(t)} \cap \{w, x\} \in \mathcal{S}^{(t)}] = p^{(t)}, \ \forall t \geq 1, \ \forall \{u, v\} \neq \{w, x\} \in \mathcal{E}^{(t)}. \quad (3)$$

Proof. See Sect. A.2 of the supplementary document [1] for a proof. ■

(Dis)advantages of ThinkD$_{\text{acc}}$: Within the same memory budget, THINKD$_{\text{ACC}}$ is slower than THINKD$_{\text{FAST}}$ since THINKD$_{\text{ACC}}$ maintains and processes more samples on average. However, THINKD$_{\text{ACC}}$ is more accurate than THINKD$_{\text{FAST}}$ by utilizing more samples. These are shown empirically in Sect. 5.3 and Sect. 5.4.

Reducing estimation errors by sacrificing unbiasedness: The estimates (i.e., $\bar{c}$ and $c[u]$ for each node u) in Algorithms 1 and 2 can have negative values. Since true triangle counts are always non-negative, lower bounding the estimates

by zero always reduces the estimation errors. However, the estimates become biased, and Theorem 1 in the following section does not hold anymore.

4.3 Accuracy Analyses

We prove that THINKD$_{\text{FAST}}$ and THINKD$_{\text{ACC}}$ maintain unbiased estimates with the expected values equal to the true global and local triangle counts. Then, we analyze the variances of the estimates that THINKD$_{\text{FAST}}$ maintains. To this end, for each variable (e.g., $\bar{c}$) in Algorithms 1 and 2, we use superscript (t) (e.g., $\bar{c}^{(t)}$) to denote the value of the variable after the t-th element $e^{(t)}$ is processed.

We first define *added triangles* and *deleted triangles* in Definitions 1 and 2.

Definition 1 (Added Triangles). *Let $\mathcal{A}^{(t)}$ be the set of triangles that have been added to graph $\mathcal{G}$ at time t or earlier. Formally,*

$$\mathcal{A}^{(t)} := \{(\{u,v,w\}, s) : 1 \leq s \leq t \text{ and } \{u,v,w\} \notin \mathcal{T}^{(s-1)} \text{ and } \{u,v,w\} \in \mathcal{T}^{(s)}\},$$

where addition time s is for distinguishing triangles composed of the same nodes but added at different times. [1]

Definition 2 (Deleted Triangles). *Let $\mathcal{D}^{(t)}$ be the set of triangles that have been removed from graph $\mathcal{G}$ at time t or earlier. Formally,*

$$\mathcal{D}^{(t)} := \{(\{u,v,w\}, s) : 1 \leq s \leq t \text{ and } \{u,v,w\} \in \mathcal{T}^{(s-1)} \text{ and } \{u,v,w\} \notin \mathcal{T}^{(s)}\},$$

where deletion time s is for distinguishing triangles composed of the same nodes but deleted at different times. [1]

Similarly, for each node $u \in \mathcal{V}^{(t)}$, we use $\mathcal{A}^{(t)}[u] \subset \mathcal{A}^{(t)}$ and $\mathcal{D}^{(t)}[u] \subset \mathcal{D}^{(t)}$ to denote the added and deleted triangles with node u, respectively. Lemma 3 formalizes the relationship between these concepts and the number of triangles.

Lemma 3 (Count of Triangles in the Current Graph). *The count of triangles in the current graph equals to the count of added triangles subtracted by the count of deleted triangles. Formally,*

$$|\mathcal{T}^{(t)}| = |\mathcal{A}^{(t)}| - |\mathcal{D}^{(t)}|, \ \forall t \geq 1, \tag{4}$$

$$|\mathcal{T}^{(t)}[u]| = |\mathcal{A}^{(t)}[u]| - |\mathcal{D}^{(t)}[u]|, \ \forall t \geq 1, \ \forall u \in \mathcal{V}^{(t)}. \tag{5}$$

Proof. Eq. (4) and Eq. (5) follow from Definitions 1 and 2. See Sect. A.3 of the supplementary document [1] for a formal proof. ■

Based on these concepts, we prove that THINKD$_{\text{FAST}}$ and THINKD$_{\text{ACC}}$ maintain unbiased estimates in Theorem 1. For the unbiasedness of the estimate $\bar{c}$ of the global count, we show that the expected amount of change in $\bar{c}$ for each added triangle is $+1$, while that for each deleted triangle is -1. Then, by Lemma 3, the expected value of $\bar{c}$ equals to the true global count. Likewise, we show the unbiasedness of the estimate of the local triangle count of each node by considering only the added and deleted triangles incident to the node.

[1] Note that triangles composed of the same nodes can be added multiple times (and thus can be removed multiple times) only if deleted edges are added again.

Theorem 1 ('Any Time' Unbiasedness of ThinkD). THINKD *gives unbiased estimates at any time. Formally, in Algorithms 1 and 2,*

$$\mathbb{E}[\bar{c}^{(t)}] = |\mathcal{T}^{(t)}|, \ \forall t \geq 1, \tag{6}$$

$$\mathbb{E}[c^{(t)}[u]] = |\mathcal{T}^{(t)}[u]|, \ \forall t \geq 1, \ \forall u \in \mathcal{V}^{(t)}. \tag{7}$$

Proof. Consider a triangle $(\{u, v, w\}, s) \in \mathcal{A}^{(t)}$, and let $e^{(s)} = (\{u, v\}, +)$ without loss of generality. The amount $\alpha^{(s)}_{uvw}$ of change in each of $\bar{c}$, $c[u]$, $c[v]$, and $c[w]$ due to the discovery of $(\{u, v, w\}, s)$ in line 8 of Algorithm 1 or Algorithm 2 is

$$\alpha^{(s)}_{uvw} = \begin{cases} 1/r^2 & if \{v, w\} \in \mathcal{S}^{(s-1)} \text{ and } \{w, u\} \in \mathcal{S}^{(s-1)} \text{ in Algorithm 1} \\ 1/p^{(s-1)} & if \{v, w\} \in \mathcal{S}^{(s-1)} \text{ and } \{w, u\} \in \mathcal{S}^{(s-1)} \text{ in Algorithm 2} \\ 0 & otherwise. \end{cases}$$

Then, from Eq. (1) and Eq. (3), the following equation holds:

$$\alpha^{(s)}_{uvw} = \begin{cases} \frac{1}{Pr[\{v,w\} \in \mathcal{S}^{(s-1)} \cap \{w,u\} \in \mathcal{S}^{(s-1)}]} & if \{v, w\} \in \mathcal{S}^{(s-1)} \text{ and } \{w, u\} \in \mathcal{S}^{(s-1)} \\ 0 & otherwise. \end{cases}$$

Hence,

$$\mathbb{E}[\alpha^{(s)}_{uvw}] = 1. \tag{8}$$

Consider a triangle $(\{u, v, w\}, s) \in \mathcal{D}^{(t)}$, and let $e^{(s)} = (\{u, v\}, -)$ without loss of generality. The amount $\beta^{(s)}_{uvw}$ of change in each of $\bar{c}$, $c[u]$, $c[v]$, and $c[w]$ due to the discovery of $(\{u, v, w\}, s)$ in line 9 of Algorithm 1 or Algorithm 2 is

$$\beta^{(s)}_{uvw} = \begin{cases} -1/r^2 & if \{v, w\} \in \mathcal{S}^{(s-1)} \text{ and } \{w, u\} \in \mathcal{S}^{(s-1)} \text{ in Algorithm 1} \\ -1/p^{(s-1)} & if \{v, w\} \in \mathcal{S}^{(s-1)} \text{ and } \{w, u\} \in \mathcal{S}^{(s-1)} \text{ in Algorithm 2} \\ 0 & otherwise. \end{cases}$$

Then, from Eq. (1) and Eq. (3), the following equation holds:

$$\beta^{(s)}_{uvw} = \begin{cases} \frac{-1}{Pr[\{v,w\} \in \mathcal{S}^{(s-1)} \cap \{w,u\} \in \mathcal{S}^{(s-1)}]} & if \{v, w\} \in \mathcal{S}^{(s-1)} \text{ and } \{w, u\} \in \mathcal{S}^{(s-1)} \\ 0 & otherwise. \end{cases}$$

Hence,

$$\mathbb{E}[\beta^{(s)}_{uvw}] = -1. \tag{9}$$

By definition, the following holds:

$$\bar{c}^{(t)} = \sum_{(\{u,v,w\},s) \in \mathcal{A}^{(t)}} \alpha^{(s)}_{uvw} + \sum_{(\{u,v,w\},s) \in \mathcal{D}^{(t)}} \beta^{(s)}_{uvw}.$$

By linearity of expectation, Eq. (8), Eq. (9), and Lemma 3, the following holds:

$$\begin{aligned} \mathbb{E}[\bar{c}^{(t)}] &= \sum_{(\{u,v,w\},s) \in \mathcal{A}^{(t)}} \mathbb{E}[\alpha^{(s)}_{uvw}] + \sum_{(\{u,v,w\},s) \in \mathcal{D}^{(t)}} \mathbb{E}[\beta^{(s)}_{uvw}] \\ &= \sum_{(\{u,v,w\},s) \in \mathcal{A}^{(t)}} 1 + \sum_{(\{u,v,w\},s) \in \mathcal{D}^{(t)}} (-1) = |\mathcal{A}^{(t)}| - |\mathcal{D}^{(t)}| = |\mathcal{T}^{(t)}|. \end{aligned}$$

Likewise, for each node $u \in \mathcal{V}^{(t)}$, the following holds:

$$c^{(t)}[u] = \sum_{(\{u,v,w\},s) \in \mathcal{A}^{(t)}[u]} \alpha^{(s)}_{uvw} + \sum_{(\{u,v,w\},s) \in \mathcal{D}^{(t)}[u]} \beta^{(s)}_{uvw}.$$

By linearity of expectation, Eq. (8), Eq. (9), and Lemma 3, the following holds:

$$\begin{aligned}\mathbb{E}[c^{(t)}[u]] &= \sum_{(\{u,v,w\},s) \in \mathcal{A}^{(t)}[u]} \mathbb{E}[\alpha^{(s)}_{uvw}] + \sum_{(\{u,v,w\},s) \in \mathcal{D}^{(t)}[u]} \mathbb{E}[\beta^{(s)}_{uvw}] \\ &= \sum_{(\{u,v,w\},s) \in \mathcal{A}^{(t)}[u]} 1 + \sum_{(\{u,v,w\},s) \in \mathcal{D}^{(t)}[u]} (-1) = |\mathcal{A}^{(t)}[u]| - |\mathcal{D}^{(t)}[u]| = |\mathcal{T}^{(t)}[u]|.\end{aligned}$$

■

In Sect. B of the supplementary document [1], we prove the formulas for the variances of estimates given by THINKD$_{\text{FAST}}$. Theorem 2 is implied by them.

Theorem 2 (Variance of ThinkD$_{\text{fast}}$). *Given an input graph stream, the variances of estimates maintained by* THINKD$_{\text{FAST}}$ *with the sampling probability r is proportional to $1/r^2$. Formally, in Algorithm 1,*

$$Var[\bar{c}^{(t)}] = O(1/r^2),\ \forall t \geq 1, \quad \textit{and} \quad Var[c^{(t)}[u]] = O(1/r^2),\ \forall t \geq 1,\ \forall u \in \mathcal{V}^{(t)}.$$

Proof. See Theorem 5 in Sect. B of the supplementary document [1]. ■

4.4 Complexity Analyses

We analyze the time and space complexities of THINKD$_{\text{FAST}}$ and THINKD$_{\text{ACC}}$. In our analyses, we use $\bar{\mathcal{V}}^{(t)} := \bigcup_{s=1}^{t} \mathcal{V}^{(s)}$ to denote the set of nodes that appear in the t-th or earlier elements in the input stream.

Space Complexity: To process the first t elements in the input graph stream, THINKD$_{\text{FAST}}$ and THINKD$_{\text{ACC}}$ maintain one estimate for the global triangle count and at most $|\bar{\mathcal{V}}^{(t)}|$ estimates for the local triangle counts. In addition, THINKD$_{\text{FAST}}$ maintains $|\mathcal{E}^{(t)}| \cdot r$ edges on average, while THINKD$_{\text{ACC}}$ maintains up to k edges. Thus, the average space complexities of THINKD$_{\text{FAST}}$ and THINKD$_{\text{ACC}}$ are $O(|\mathcal{E}^{(t)}| \cdot r + |\bar{\mathcal{V}}^{(t)}|)$ and $O(k + |\bar{\mathcal{V}}^{(t)}|)$, respectively. The complexities become $O(|\mathcal{E}^{(t)}| \cdot r)$ and $O(k)$ when only the global triangle count needs to be estimated.

Time Complexity: We prove the average time complexity of THINKD$_{\text{FAST}}$ in Theorem 3, which implies Corollary 1, and the worst-case time complexity of THINKD$_{\text{ACC}}$ in Theorem 4. Corollary 1 and Theorem 4 state that, given a fixed memory budget k, THINKD$_{\text{FAST}}$ and THINKD$_{\text{ACC}}$ scale linearly with the number of elements in the input stream.

Theorem 3 (Time Complexity of ThinkD$_{\text{fast}}$). *Algorithm 1 takes $O(t + t^2 r)$ on average to process the first t elements in the input stream.*

Table 3: Summary of the real-world and synthetic graph streams used in our experiments. B: billion, M: million, K: thousand.

Name	#Nodes	#Edges	Type	Name	#Nodes	#Edges	Type
Friendster	65.6M	1.81B	Friendship	Youtube	3.22M	9.38M	Friendship
Orkut	3.07M	117M	Friendship	BerkStan	685K	6.65M	Web
Flickr	2.30M	22.8M	Friendship	Facebook	63.7K	817K	Friendship
Patent	3.77M	16.5M	Citation	Epinion	132K	711K	Trust
Random (800GB)	1M	0.1B-100B	Synthetic				

Proof. In Algorithm 1, the most expensive step in processing each element $e^{(s)} = (\{u, v\}, \delta)$ is to intersect $\hat{\mathcal{N}}[u]$ and $\hat{\mathcal{N}}[v]$ (line 7), which takes $O(1 + \mathbb{E}[|\hat{\mathcal{N}}[u]| + |\hat{\mathcal{N}}[v]|]) = O(1 + \mathbb{E}[|\mathcal{S}|]) = O(1 + sr)$ on average. Hence, processing the first t elements takes $\sum_{s=1}^{t} O(1 + sr) = O(t + t^2 r)$ on average. ■

Corollary 1 (Time Complexity of ThinkD$_{\text{fast}}$ with Fixed Memory k). *If $r = O(k/t)$ for a constant k (≥ 1), then Algorithm 1 takes $O(tk)$ on average to process the first t elements in the input stream.*

Theorem 4 (Time Complexity of ThinkD$_{\text{acc}}$). *Algorithm 2 takes $O(tk)$ to process the first t elements in the input stream.*

Proof. In Algorithm 2, the most expensive step in processing each element $e^{(s)} = (\{u, v\}, \delta)$ is to intersect $\hat{\mathcal{N}}[u]$ and $\hat{\mathcal{N}}[v]$ (line 7), which takes $O(1 + |\hat{\mathcal{N}}[u]| + |\hat{\mathcal{N}}[v]|) = O(k)$. Thus, processing the first t elements takes $O(tk)$. ■

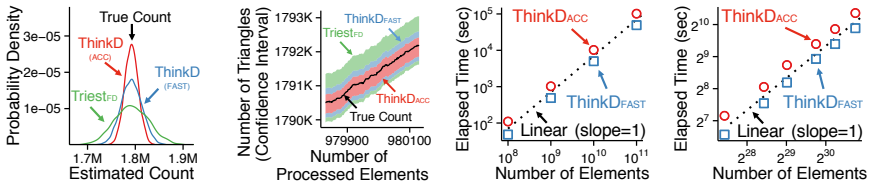
5 Experiments

In this section, we review our experiments for answering the following questions:

- **Q1. Illustration of Theorems**: Does ThinkD give unbiased estimates? Does ThinkD scale linearly with the size of the input stream?
- **Q2. Accuracy**: Is ThinkD more accurate than its best competitors?
- **Q3. Speed**: Is ThinkD faster than its best competitors?
- **Q4. Effects of Deletions**: Is ThinkD consistently accurate regardless of the ratio of deleted edges?

5.1 Experimental Settings

Machines: We used a machine with a 3.60GHz CPU and 32GB RAM unless otherwise stated.
Datasets: We created fully dynamic graph streams with deletions using the real-world graphs listed in Table 3 as follows: (a) create the additions of the edges in the input graph and shuffle them, (b) choose $\alpha\%$ of the edges and create the deletions of them, (c) locate each deletion in a random position after the



(a) Distribution of estimates at the end (b) 95% confidence intervals over time (c) Scalability (Random) (d) Scalability (Friendster)

Fig. 1: **ThinkD is provably accurate and scalable.** (a) THINKD gives unbiased estimates with smaller variances than its best competitor. (b) THINKD maintains more accurate estimates with smaller confidence intervals than its best competitor. (c-d) THINKD scales linearly with the size of the input stream.

corresponding addition. We set α to 20% unless otherwise stated (see Sect. 5.5 for its effect on accuracy). The created streams were streamed from the disk.

Implementations: We implemented THINKD$_{\text{FAST}}$, THINKD$_{\text{ACC}}$, TRIEST$_{\text{FD}}$ [7], TRIEST$_{\text{IMPR}}$ [7], ESD [10], and MASCOT [14] in Java 1.7. In all of them, sampled edges are stored in the adjacency list format, and as described in the last paragraph of Sect. 4.2, estimates are lower bounded by zero.

Evaluation Metrics: Let x and $\{(u, x[u])\}_{u\in\mathcal{V}}$ be the true counts of global triangles and local triangles at the end of the input stream. Let $\hat{x}$ and $\{(u, \hat{x}[u])\}_{u\in\mathcal{V}}$ be the corresponding estimates obtained by the evaluated algorithm. We used *global error*, defined as $\frac{|x-\hat{x}|}{1+x}$, and *RMSE*, defined as $\sqrt{\frac{1}{|\mathcal{V}|}\sum_{u\in\mathcal{V}}(x[u]-\hat{x}[u])^2}$, to evaluate the accuracy of global and local triangle counting, respectively.

5.2 Q1. Illustration of Theorems

ThinkD gives unbiased estimates (Theorem 1). We compared 10,000 estimates of the global triangle count obtained by THINKD$_{\text{FAST}}$, THINKD$_{\text{ACC}}$, and TRIEST$_{\text{FD}}$, whose parameters were set so that on average 10% of the edges are stored at the end of each graph stream. Figure 1(a) shows the distributions of the estimates at the end of the Facebook dataset. The means of the estimates were close to the true triangle count, consistently with Theorem 1 (i.e., unbiasedness of THINKD). Moreover, THINKD$_{\text{ACC}}$ and THINKD$_{\text{FAST}}$ gave estimates with smaller variances than TRIEST$_{\text{FD}}$. Figure 1(b) shows how the 95% confidence intervals change over time in the Facebook dataset. THINKD$_{\text{FAST}}$ and THINKD$_{\text{ACC}}$ maintained more accurate estimates with smaller confidence intervals than TRIEST$_{\text{FD}}$. Between THINKD$_{\text{FAST}}$ and THINKD$_{\text{ACC}}$, THINKD$_{\text{ACC}}$ was more accurate.

ThinkD scales linearly (Corollary 1 and Theorem 4). We measured the elapsed times taken by THINKD$_{\text{FAST}}$ and THINKD$_{\text{ACC}}$ to process all elements in graph streams with different numbers of elements. To measure their speeds independently of the speed of the input stream, we ignored time taken to wait for the arrival of elements. In both algorithms, we set k and r so that on average 10^7

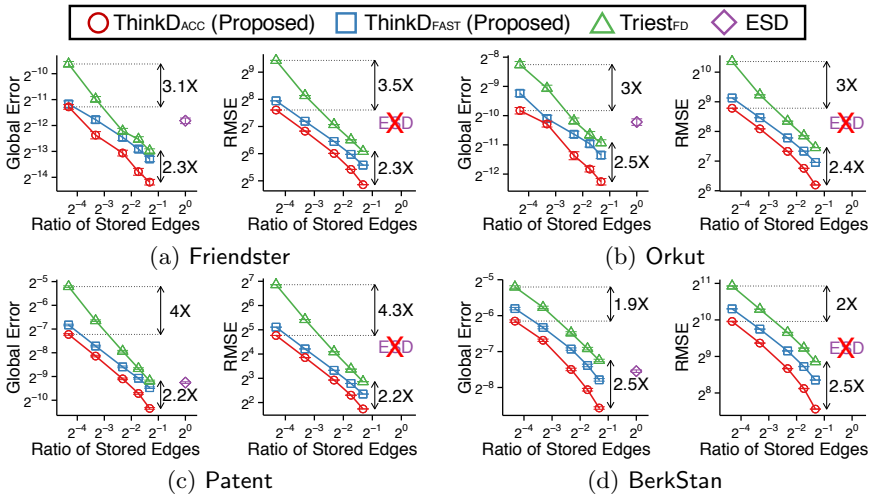


Fig. 2: **ThinkD is accurate**. ThinkD gives the best trade-off between space and accuracy. In particular, ThinkD$_{\text{ACC}}$ is up to **4.3× more accurate** than Triest$_{\text{FD}}$ within the same memory budget. Error bars denote ±1 standard error. ESD is inapplicable to local triangle counting.

edges are stored at the end of each input stream. Figure 1(c) shows the results in the Random datasets, which were created by the Erdös-Rényi model. Both ThinkD$_{\text{FAST}}$ and ThinkD$_{\text{ACC}}$ scaled linearly with the number of elements, as expected in Corollary 1 and Theorem 4. Notice that the largest dataset is **800GB with 100 billion elements**. As seen in Fig. 1(d), ThinkD$_{\text{FAST}}$ and ThinkD$_{\text{ACC}}$ showed linear scalability also in a graph stream with realistic structure, which we created by sampling different numbers of elements from the Friendster dataset.

5.3 Q2. Accuracy (ThinkD is more accurate than its competitors)

We compared the accuracies of four algorithms that support edge deletions. As we changed the ratio of stored edges at the end of each input stream from 5% to 40%, we measured the accuracies of ThinkD$_{\text{FAST}}$, ThinkD$_{\text{ACC}}$, and Triest$_{\text{FD}}$. ESD always stores the entire input stream in memory, and we set its parameter to 1.0 to maximize its accuracy. Each evaluation metric was averaged over 100 trials in the Friendster and Orkut datasets and 1,000 trials in the others.[2] As seen in Fig. 2, ThinkD$_{\text{FAST}}$ and ThinkD$_{\text{ACC}}$ consistently gave the best trade-off between space and accuracy. Specifically, within the same memory budget, ThinkD$_{\text{ACC}}$ was up to **4**× and **4.3**× **more accurate** than Triest$_{\text{FD}}$ in terms of global error and RMSE, respectively. Between our algorithms, ThinkD$_{\text{ACC}}$ consistently outperformed ThinkD$_{\text{FAST}}$. We observed the same trend in the other datasets (see Fig. 5 in the supplementary document [1]).

[2] We used a machine with 2.67GHz CPUs and 1TB memory for the Friendster dataset.

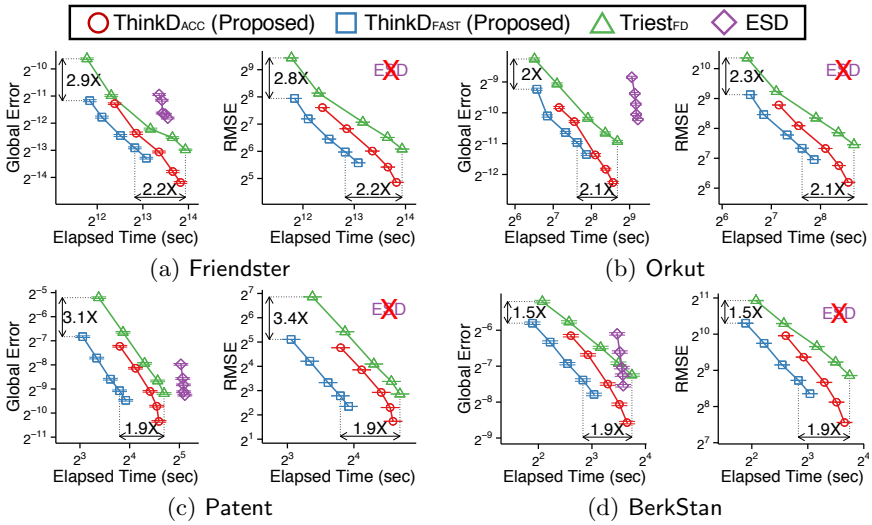


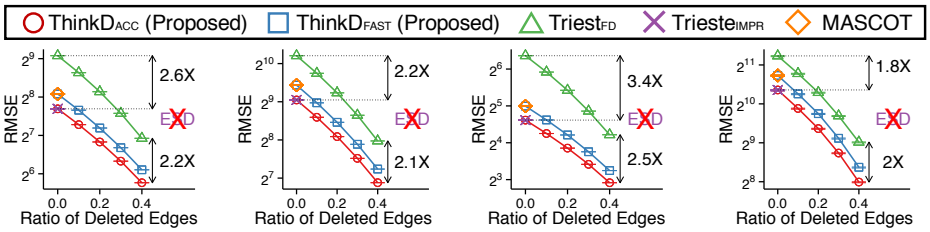
Fig. 3: **ThinkD is fast**. ThinkD gives the best trade-off between speed and accuracy. In particular, ThinkD$_{\text{FAST}}$ is up to **2.2× faster** than Triest$_{\text{FD}}$ when they are similarly accurate. Error bars denote ±1 standard error. ESD is inapplicable to local triangle counting.

5.4 Q3. Speed (ThinkD is faster than its competitors)

We compared the speeds and accuracies of four algorithms that support edge deletions. The detailed settings were the same as those in Sect. 5.3 except that we measured the performance of ESD as we changed its parameter from 0.2 to 1.0. To measure the speeds of the algorithms independently of the speed of the input stream, we ignored time taken to wait for the arrival of elements. As seen in Fig. 3, ThinkD$_{\text{FAST}}$ and ThinkD$_{\text{ACC}}$ consistently gave the best trade-off between speed and accuracy. Specifically, for the same global error and RMSE, ThinkD$_{\text{FAST}}$ was up to **2.2× faster** than Triest$_{\text{FD}}$. Between our algorithms, ThinkD$_{\text{FAST}}$ consistently outperformed ThinkD$_{\text{ACC}}$. We observed the same trend in the other datasets (see Fig. 6 in the supplementary document [1]).

5.5 Q4. Effects of Deletions (ThinkD is consistently accurate)

We measured how the ratio of deleted edges (i.e., α in Sect. 5.1) in input graph streams affects the accuracies of the considered algorithms. In every algorithm, we set the ratio of stored edges at the end of each input stream to 10%. As seen in Fig. 4, all algorithms that support edge deletions became more accurate as input graphs became smaller with more deletions. ThinkD$_{\text{FAST}}$ and ThinkD$_{\text{ACC}}$ were similarly accurate with Mascot and Triest$_{\text{IMPR}}$, respectively, in the streams without deletions. In the streams with deletions, which Mascot and Triest$_{\text{IMPR}}$ cannot handle, ThinkD$_{\text{FAST}}$ and ThinkD$_{\text{ACC}}$ were **1.8 − 3.4× more accurate**



(a) Friendster (b) Orkut (c) Patent (d) BerkStan

Fig. 4: **ThinkD is consistently accurate regardless of the ratio of deleted edges.** Error bars denote ±1 standard error. TRIEST$_{\text{IMPR}}$ and MASCOT are inapplicable when there are deletions. ESD is inapplicable to local triangle counting.

than TRIEST$_{\text{FD}}$ regardless of the ratio of deleted edges. We observed the same trend in the other datasets (see Fig. 7 in the supplementary document [1]).

6 Conclusion

We propose THINKD, which estimates the counts of global and local triangles in a fully dynamic graph stream with edge additions and deletions. Our theoretical and empirical analyses show that THINKD has the following advantages:

- **Accurate**: THINKD is up to *4.3× more accurate* than its best competitors within the same memory budget (Fig. 2).
- **Fast**: THINKD is up to *2.2× faster* than its best competitors with similar accuracies (Fig. 3). THINKD processes *terabyte*-scale graph streams with linear scalability (Fig. 1, Corollary 1, and Theorem 4).
- **Theoretically Sound**: THINKD maintains *unbiased* estimates (Theorem 1) with small variances (Theorem 2) *at any time* while the input graph evolves.

Reproducibility: The source code and datasets used in the paper are available at `http://www.cs.cmu.edu/~kijungs/codes/thinkd/`.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grants No. CNS-1314632 and IIS-1408924. Research was sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-09-2-0053. Shin was supported by the KFAS Scholarship, and Kim was supported by the Samsung Scholarship. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation, or other funding parties. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

References

1. Supplementary document. Available online: `http://www.cs.cmu.edu/~kijungs/codes/thinkd/supple.pdf` (2018)
2. Ahmed, N.K., Duffield, N., Willke, T.L., Rossi, R.A.: On sampling from massive graph streams. PVLDB 10(11), 1430–1441 (2017)
3. Bar-Yossef, Z., Kumar, R., Sivakumar, D.: Reductions in streaming algorithms, with an application to counting triangles in graphs. In: SODA (2002)
4. Batagelj, V., Zaveršnik, M.: Short cycle connectivity. Discrete Mathematics 307(3), 310–318 (2007)
5. Becchetti, L., Boldi, P., Castillo, C., Gionis, A.: Efficient algorithms for large-scale local triangle counting. TKDD 4(3), 13 (2010)
6. Berry, J.W., Hendrickson, B., LaViolette, R.A., Phillips, C.A.: Tolerating the community detection resolution limit with edge weighting. Physical Review E 83(5), 056119 (2011)
7. De Stefani, L., Epasto, A., Riondato, M., Upfal, E.: Trièst: Counting local and global triangles in fully-dynamic streams with fixed memory size. In: KDD (2016)
8. Epasto, A., Lattanzi, S., Mirrokni, V., Sebe, I.O., Taei, A., Verma, S.: Ego-net community mining applied to friend suggestion. PVLDB 9(4), 324–335 (2015)
9. Gemulla, R., Lehner, W., Haas, P.J.: Maintaining bounded-size sample synopses of evolving datasets. The VLDB Journal 17(2), 173–201 (2008)
10. Han, G., Sethu, H.: Edge sample and discard: A new algorithm for counting triangles in large dynamic graphs. In: ASONAM (2017)
11. Jha, M., Seshadhri, C., Pinar, A.: A space efficient streaming algorithm for triangle counting using the birthday paradox. In: KDD (2013)
12. Kolountzakis, M.N., Miller, G.L., Peng, R., Tsourakakis, C.E.: Efficient triangle counting in large graphs via degree-based vertex partitioning. In: WAW (2010)
13. Kutzkov, K., Pagh, R.: Triangle counting in dynamic graph streams. In: SWAT (2014)
14. Lim, Y., Kang, U.: Mascot: Memory-efficient and accurate sampling for counting local triangles in graph streams. In: KDD (2015)
15. Newman, M.E.: The structure and function of complex networks. SIAM review 45(2), 167–256 (2003)
16. Pavan, A., Tangwongsan, K., Tirthapura, S., Wu, K.L.: Counting and sampling triangles from a graph stream. PVLDB 6(14), 1870–1881 (2013)
17. Shin, K.: Wrs: Waiting room sampling for accurate triangle counting in real graph streams. In: ICDM (2017)
18. Shin, K., Eliassi-Rad, T., Faloutsos, C.: Patterns and anomalies in k-cores of real-world graphs with applications. Knowl. Inf. Syst. 54(3), 677–710 (2018)
19. Shin, K., Hammoud, M., Lee, E., Oh, J., Faloutsos, C.: Tri-fly: Distributed estimation of global and local triangle counts in graph streams. In: PAKDD (2018)
20. Tangwongsan, K., Pavan, A., Tirthapura, S.: Parallel triangle counting in massive streaming graphs. In: CIKM (2013)
21. Tsourakakis, C.E.: Fast counting of triangles in large real networks without counting: Algorithms and laws. In: ICDM (2008)
22. Tsourakakis, C.E., Drineas, P., Michelakis, E., Koutis, I., Faloutsos, C.: Spectral counting of triangles via element-wise sparsification and triangle-based link recommendation. Social Network Analysis and Mining 1(2), 75–81 (2011)
23. Vitter, J.S.: Random sampling with a reservoir. TOMS 11(1), 37–57 (1985)
24. Watts, D.J., Strogatz, S.H.: Collective dynamics of 'small-world' networks. Nature 393(6684), 440–442 (1998)