Fast, Accurate and Provable Triangle Counting in Fully Dynamic Graph Streams

KIJUNG SHIN, KAIST

SEJOON OH, Georgia Institute of Technology

JISU KIM, Inria Saclay

BRYAN HOOI, National University of Singapore

CHRISTOS FALOUTSOS, Carnegie Mellon University

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 additions and deletions of edges. Compared to its best competitors, THINKD is **(a) Accurate:** up to $4.3\times$ more accurate within the same memory budget, **(b) Fast:** up to $2.2\times$ faster for the same accuracy requirements, and **(c) Theoretically sound:** always maintaining estimates with zero bias (i.e., the difference between the true triangle count and the expected value of its estimate) and small variance. As an application, we use THINKD to detect suddenly emerging dense subgraphs, and we show its advantages over state-of-the-art methods.

CCS Concepts: • **Information systems** → **Data mining**; • **Theory of computation** → **Dynamic graph algorithms**; **Streaming, sublinear and near linear time algorithms**;

Additional Key Words and Phrases: Triangle counting, local triangles, edge deletions

This work was partly supported by National Research Foundation of Korea (NRF) grant funded by the Korea government MSIT (No. NRF-2019R1F1A1059755) and Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government MSIT (No. 2019-0-00075-001, Artificial Intelligence Graduate School Program (KAIST)). This work was supported by the National Science Foundation under Grant 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.

Authors' addresses: K. Shin, Graduate School of AI & School of Electrical Engineering, KAIST, 291 Daehak-ro, Yuseong-gu, Daejeon 34141, Republic of Korea; email: kijungs@kaist.ac.kr; S. Oh, School of Computational Science and Engineering, Georgia Institute of Technology, 756 W Peachtree St NW, Atlanta, GA 30308; email: soh337@gatech.edu; J. Kim, Equipe DataShape, Inria Saclay, 1 Rue Honoré d'Estienne d'Orves, Bâtiment Alan Turing, Campus de l'École Polytechnique, Palaiseau, Île-De-France 91120, France; email: jisu.kim@inria.fr; B. Hooi, Department of Computer Science, National University of Singapore, COM1, 13 Computing Drive, Singapore 117417, Republic of Singapore; email: bhooi@comp.nus.edu.sg; C. Faloutsos, Computer Science Department, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213-3891; email: christos@cs.cmu.edu.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2020 Association for Computing Machinery.

1556-4681/2020/02-ART12 \$15.00

<https://doi.org/10.1145/3375392>

ACM Reference format:

Kijung Shin, Sejoon Oh, Jisu Kim, Bryan Hooi, and Christos Faloutsos. 2020. Fast, Accurate and Provable Triangle Counting in Fully Dynamic Graph Streams. *ACM Trans. Knowl. Discov. Data* 14, 2, Article 12 (February 2020), 39 pages.

<https://doi.org/10.1145/3375392>

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 [Becchetti et al. 2010; Lim et al. 2018], link recommendation [Epasto et al. 2015b; Tsourakakis et al. 2011], community detection [Berry et al. 2011], degeneracy estimation [Shin et al. 2018a], and query optimization [Bar-Yossef et al. 2002]. In particular, many important metrics in social network analysis, including the clustering coefficient [Watts and Strogatz 1998], the transitivity ratio [Newman 2003], and the triangle connectivity [Batagelj and Zaveršnik 2007], are based on the count of triangles. We refer interested readers to [Chu and Cheng 2012; Hu et al. 2014; Kolountzakis et al. 2010; Suri and Vassilvitskii 2011] for detailed descriptions of the applications.

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.

As a result, there has been great interest in streaming 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 [Ahmed et al. 2017; Jha et al. 2013; Lim et al. 2018; Pavan et al. 2013; Shin et al. 2018b; Tangwongsan et al. 2013] or greatly sacrifice accuracy to support edge deletions [De Stefani et al. 2017; Han and Sethu 2017; Kutzkov and Pagh 2014].

Why is it challenging to accurately estimate the count of triangles while handling deletions? Most existing streaming algorithms for triangle counting are randomized algorithms, whose outputs are random variables. Their accuracy depends on the bias and variance of the random variables, which are estimates of the triangle counts. State-of-the-art algorithms [De Stefani et al. 2017; Lim et al. 2018; Shin 2017; Shin et al. 2018b] discover a subset of triangles, within a fixed memory budget, and obtain unbiased estimates¹ by dividing the size of the subset by the probability that each triangle is discovered during the process. To obtain unbiased estimates, the discovery probability needs to be computed exactly, while to reduce variances, the discovery probability needs to be increased. Instead of using only the edges stored in memory, the aforementioned algorithms, which are for triangle counting in insertion-only streams, uses every arrived edge to improve its estimation, even if the edge is about to be discarded without being stored. While this idea of utilizing edges to be discarded significantly increases the discovery probability and thus reduces the variance of estimates, it has not been applied to triangle counting in graph streams with edge deletions, for which thus state-of-the-art algorithms [De Stefani et al. 2017] suffer from low accuracy. A major challenge of the application is to compute the exact discovery probability, which is necessary to obtain unbiased estimates.

¹That is, the expected values of the estimates are equal to the true triangle counts.

Can we accurately estimate the count of triangles in fully dynamic graph streams with edge additions and deletions? In this work,² we propose THINKD (**Think** before you **Discard**), an accurate streaming algorithm for global and local triangle counting in such a graph stream. That is, THINKD maintains and updates estimates of the counts of global triangles (i.e., all triangles) and local triangles incident to each node. For this problem, THINKD is the first algorithm that utilizes edges to be discarded. We first derive formulas of the exact probability that each added or deleted triangle is discovered when edges to be discarded are used with the following two different sampling schemes: Bernoulli trials and Random Pairing (RP) [Gemulla et al. 2008]. Then, based on the formulas, THINKD utilizes edges to be discarded while guaranteeing the unbiasedness of its estimates. We formally prove that THINKD_{FAST} and THINKD_{ACC}, which are two versions of THINKD based on Bernoulli trials and RP, respectively, produce unbiased estimates. We also prove a formula of the variance of estimates given by THINKD_{FAST}; and the time and space complexities of THINKD_{FAST} and THINKD_{ACC}. Additionally, through extensive experiments using eight real-world graphs, we demonstrate that the estimates have smaller variances and thus smaller errors than estimates obtained by state-of-the-art competitors, which fail to utilize edges to be discarded. Specifically, 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 (Figure 4).
- **Fast:** THINKD *scales linearly* with the size of the input stream (Figure 3, Corollary 1, and Theorem 4). Especially, THINKD is up to 2.2× *faster* than its best competitors with similar accuracies.
- **Theoretically Sound:** We prove the formulas for the bias and variance of the estimates given by THINKD (Theorems 1 and 2). In particular, we show that THINKD always maintains *unbiased* estimates with small variances (Figure 2).

Additionally, as a new application, we use THINKD to detect suddenly emerging dense subgraphs, and we show its advantages over state-of-the-art methods.

Reproducibility: The source code and datasets used in the article are available at <http://dmlab.kaist.ac.kr/~kijungs/codes/thinkd/>.

The rest of the paper is organized as follows. In Section 2, we review related work. In Section 3, we introduce notations and the problem definition. In Section 4, we describe our proposed algorithm THINKD and analyze its accuracy and complexity. After providing experimental results in Section 5, we conclude in Section 6. In Appendix A, we present a toy example that highlights (dis)advantages of THINKD and its competitors. In Appendix B, we present a theoretical analysis of the variances of estimates given by THINKD. In Appendix C, we extend THINKD to triangle counting in fully dynamic multigraph streams with parallel edges. In Appendix D, we apply THINKD to the task of detecting suddenly emerging dense subgraphs.

2 RELATED WORK

We review previous work on triangle counting in insertion-only or fully-dynamic graph streams. See Table 1 for a comparison of streaming algorithms for triangle counting.

²This work is an extended version of [Shin et al. 2018d] with proofs, a variance analysis, additional experimental results, an extension of THINKD for multigraph streams, and an application of THINKD to detecting suddenly emerging dense subgraphs.

Table 1. Comparison of Algorithms for Triangle Counting in Graph Streams

	THINKD (Proposed)	TRIEST _{FD} [De Stefani et al. 2017]	ESD [Han and Sethu 2017]	MASCOT [Lim et al. 2018]	TRIEST _{IMPR} [De Stefani et al. 2017]	WRS [Shin 2017]	FURL [Jung et al. 2016]	PARTITIONCT [Wang et al. 2017]	Others*
Counting Global Triangles	✓	✓	✓	✓	✓	✓		✓	✓
Counting Local Triangles	✓	✓		✓	✓	✓	✓		
Handling Large Graphs**	✓	✓		✓	✓	✓	✓	✓	✓
Handling Edge Insertions	✓	✓	✓	✓	✓	✓	✓	✓	✓
Handling Edge Deletions	✓	✓	✓						
Handling Parallel Edges	✓	✓		✓	✓		✓	✓	
Guaranteeing Unbiasedness***	✓	✓	✓	✓	✓	✓		✓	✓

Note that THINKD satisfies all the criteria while clearly outperforming TRIEST_{FD} (which also satisfies all the criteria) in terms of speed and accuracy.

*[Ahmed et al. 2017, 2014; Tangwongsan et al. 2013; Jha et al. 2013; Pavan et al. 2013].

**graphs that do not fit in memory.

***giving estimates whose expected values are equal to the true triangle counts.

2.1 Triangle Counting in Insertion-only Graph Streams

In DOULION [Tsourakakis et al. 2009], each edge in the input graph is sampled independently with probability r . Then, the probability that all three edges in a triangle are sampled and thus discovered is r^3 . Based on this fact, DOULION provides $1/r^3$ times the triangle count in the graph composed of the sampled edges as an unbiased estimate of the triangle count in the entire graph.³ As in DOULION, once the probability that each triangle is discovered (i.e., sampled) is known, an unbiased estimate is easily obtained. Since the variance of such unbiased estimates is roughly inversely proportional to the probability that each triangle is discovered,⁴ many studies have focused on increasing the probability to reduce such variance. COLORFUL TRIANGLE SAMPLING [Pagh and Tsourakakis 2012] increases the probability to r^2 in a non-streaming setting by (a) coloring each node in the input graph with a color chosen uniformly at random among $1/r$ colors and (b) sampling the edges connecting nodes with the same color. MASCOT [Lim et al. 2018] increases the probability to r^2 in insertion-only graph streams by (a) sampling each incoming edge independently with probability r but (b) discovering (i.e., counting) triangles with each incoming edge before sampling or discarding it. Note that, in MASCOT, each triangle is discovered if and only if the first two edges of the triangle are sampled (with probability r^2). TRIEST_{IMPR} [De Stefani et al. 2017] further increases the probability in insertion-only graph streams by sampling edges using the reservoir sampling [Vitter 1985], which uniformly samples as many edges as possible within a given memory budget, while MASCOT may discard edges even when memory is underutilized. Other approaches for increasing the probability include (a) sampling wedges (i.e., paths of length two) in addition to edges

³The expected value of an unbiased estimate is equal to the true value.

⁴For example, in THINKD_{FAST}, each triangle is discovered with probability r^2 in Lemma 5 and the variance of the estimate is $O(1/r^2)$ in Theorem 2.

[Jha et al. 2013; Pavan et al. 2013; Tangwongsan et al. 2013] and (b) sampling edges with different probabilities that depend on the counts of incident triangles and adjacent sampled edges [Ahmed et al. 2014, 2017].

In addition, regarding triangle counting in insertion-only streams, handling duplicated edges [Wang et al. 2017]⁵, exploiting temporal dependencies of edges [Shin 2017], utilizing a computer cluster [Shin et al. 2018b], reducing variances of estimates by combining past estimates with the current one (at the expense of losing unbiasedness) [Jung et al. 2016], and semi-streaming algorithms requiring multiple passes over data [Kolountzakis et al. 2010; Tsourakakis 2008] were discussed.

2.2 Triangle Counting in Fully-dynamic Graph Streams

The first algorithm for triangle counting in fully dynamic graph streams with edge deletions was proposed in [Kutskov and Pagh 2014]. This algorithm adapts COLORFUL TRIANGLE SAMPLING, described in the previous subsection, to obtain sparsified graphs in a fully dynamic graph stream. Then, the ratio of two-paths that are completed to triangles is estimated from these graphs. This estimated ratio is multiplied with an estimate of the total number of two-paths in the input graph to estimate the total number of triangles. This algorithm, however, 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. Moreover, in the worst case, the algorithm requires more memory than what is needed to store the entire input graph, as pointed out in [De Stefani et al. 2017]. ESD [Han and Sethu 2017] maintains the current snapshot of the input graph, which is given as a fully dynamic graph stream. For each change $(\{u, v\}, \pm)$ in the input graph, Edge Sample and Discard (ESD) tosses a (biased) coin. If the head comes up, ESD estimates the changes in the triangle count, instead of exactly computing them, by checking whether a random neighbor of node u (or v) is also a neighbor of node v (or u), and updates its estimate of the triangle count. Otherwise, ESD does not update its estimate of the triangle count.

Unlike the previous algorithm, ESD always maintains its estimate of the triangle count. However, its scalability is limited since ESD maintains the entire input graph in memory. TRIEST_{FD} [De Stefani et al. 2017] maintains edges uniformly sampled within a given memory budget in a fully dynamic graph by employing RP [Gemulla et al. 2008] (see Section 4.1). TRIEST_{FD} also maintains unbiased estimates of global and local triangle counts in the input graph, which are obtained by multiplying (a) the triangle counts in the graph consisting of the sampled edges and (b) the reciprocal of the probability that each triangle is sampled. While TRIEST_{FD} simply discards unsampled edges without utilizing them to update its estimates, our proposed algorithm, THINKD, utilizes these unsampled edges to update estimates before discarding them to minimize information loss, and thus it obtains more accurate estimates than TRIEST_{FD}. Although this idea of using unsampled edges was first used in MASCOT [Lim et al. 2018] and TRIEST_{IMPR} [De Stefani et al. 2017], which are described in the previous subsection, for triangle counting in insertion-only streams, applying the idea to fully dynamic graph streams has remained unexplored.

In addition, semi-streaming algorithms, which require multiple passes over data, were developed for triangle counting in fully dynamic graph streams [Becchetti et al. 2010].

3 NOTATIONS AND PROBLEM DEFINITION

In this section, we first introduce notations and concepts. Then, we formally define the problem of triangle counting in fully-dynamic graph streams.

⁵Unlike in Appendix C, [Wang et al. 2017] aims to count triangles while ignoring duplicated edges.

Table 2. Table of Frequently-used Symbols

	Symbol	Definition
Notations for Fully Dynamic Graph Streams (Section 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 $\mathcal{G}^{(t)}$
Notations for Algorithms and Analyses (Section 4)	$\mathcal{T}^{(t)}[u]$	set of local triangles of node u in $\mathcal{G}^{(t)}$
	$\mathcal{S}$	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 _{FAST}
	k	maximum number of sampled edges in THINKD _{ACC}
	$\mathcal{A}^{(t)}$	set of added triangles at time t
	$\mathcal{D}^{(t)}$	set of deleted triangles at time t

3.1 Notations

Table 2 lists the symbols frequently used in the article. 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 that 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 $(e^{(1)}, e^{(2)}, \dots)$. For each $t \in \{1, 2, \dots\}$, 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 $\{u, v\} \in \mathcal{E}$. That is, we assume that only new edges can be added (see Appendix C for triangle counting in a multigraph stream with parallel edges), and only existing edges can be deleted. We use $\mathcal{G}^{(t)} = (\mathcal{V}^{(t)}, \mathcal{E}^{(t)})$ to indicate $\mathcal{G}$ at time t . That is,

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

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 $\mathcal{G}^{(t)}$.

3.2 Problem Definition

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 changes 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 $(e^{(1)}, e^{(2)}, \dots)$
(i.e., sequence of edge additions and deletions in graph $\mathcal{G}$)

(a) TRIEST-FD [De Stefani et al. 2017]

(b) ThinkD (Proposed):

Fig. 1. Description of THINKD and its best competitor TRIEST_{FD}, which support global and local triangle counting in a fully dynamic graph stream. (a) In TRIEST_{FD}, each incoming change goes through a test step (colored blue) and then an update step (colored green). The changes not passing the test step are simply discarded (see red arrows) without being utilized to update estimates. Due to this information loss, TRIEST_{FD} produces inaccurate estimates. (b) However, in THINKD (proposed), each incoming change goes through the update step (colored green) and then a test step. This guarantees that every change is used to update estimates (see the red arrow). THINKD_{FAST} and THINKD_{ACC} are distinguished by their testing scheme: tossing a biased coin or RP (Section 4.1).

- **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 $t \in \{1, 2, \dots\}$
- **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 **Discard**), 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, as illustrated in Figure 1. Specifically, whenever each change in the input stream arrives, THINKD first updates its estimates using the change and previously sampled edges as follows:

- For each observed triangle addition, increase corresponding estimates by the reciprocal of the probability that the triangle addition is observed.
- For each observed triangle deletion, decrease corresponding estimates by the reciprocal of the probability that the triangle deletion is observed.

After that, if the change is an addition of an edge, THINKD decides whether to sample the edge or not.

Estimates provided by THINKD are random variables, and its estimation error depends on the bias (i.e., the difference between the true triangle count and the expected value of its estimate) and variance of the estimates. Compared to using only sampled edges, utilizing unsampled edges increases the probability that each triangle addition or deletion is observed and thus decreases the variance of estimates, which is roughly inversely proportional to the probability (see Theorem 2). Moreover, increasing or decreasing estimates by the reciprocal of the probability, for each triangle addition or deletion, makes the biases of estimates zero and thus enables THINKD provides unbiased estimates, whose expected values are equal to the true triangle counts (see Theorem 1).

In this section, we present the following two versions of THINKD: THINKD_{FAST} and THINKD_{ACC}. THINKD_{FAST} uses a simple and fast edge sampling scheme based on Bernoulli trials, while THINKD_{ACC} is based on RP, an advanced sampling scheme for fully utilizing memory within a given budget.⁶ Specifically, we focus on (a) the detailed procedures of both versions of THINKD and (b) the probabilities that each triangle is discovered during the procedures. Then, based on the probabilities, we prove the unbiasedness of both versions of THINKD and provide a variance analysis. Next, we analyze their time and space complexities. Throughout this section, 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 $\mathcal{S}$. For each variable (e.g., $\bar{c}$), 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 by THINKD.

4.1 Preliminaries: Random Pairing

We first introduce RP [Gemulla et al. 2008], a sampling method that THINKD is based on. Then, we prove its several properties.

RP is a uniform random sampling scheme that maintains a bounded sample size under an arbitrary sequence of insertions and deletions. The dataset $\mathcal{R}$ of interest is understood as a finite set of distinguishable items, and the sample $\mathcal{S}$ is understood as a subset of $\mathcal{R}$. The dataset $\mathcal{R}$ is initially empty, and evolves over time as items are inserted and deleted. In general, items that are deleted may be subsequently re-inserted. Following [Brown and Haas 2006], a sampling scheme is called uniform if the scheme producing the sample $\mathcal{S}$ from the dataset $\mathcal{R}$ satisfies that,

$$\Pr[\mathcal{S} = \mathcal{A}] = \Pr[\mathcal{S} = \mathcal{B}], \forall \mathcal{A} \neq \mathcal{B} \subset \mathcal{R} \text{ s.t. } |\mathcal{A}| = |\mathcal{B}|. \quad (1)$$

RP is a uniform random sampling scheme with maintaining a bounded sample size, i.e., given the sample size bound $k > 0$, $|\mathcal{S}| \leq k$ always holds.

To efficiently utilize the bounded memory space, RP keeps two counters for “uncompensated” deletions. First, the counter n_b enumerates “bad” uncompensated deletions where the sample included the deleted item so that the sample size was decremented by 1 for the deletion. Second, the counter n_g enumerates “good” uncompensated deletions where the sample excluded the deleted item so that the sample size remained the same for the deletion. Obviously, $n_b + n_g$ represents the total uncompensated deletions.

RP is described in Algorithm 1. Initially, both the dataset and the sample are empty (line 2). For the deletion, if the sample includes the deleted item, RP removes the item from the sample and increments n_b (line 13). If the sample excludes the deleted item, RP increments n_g (line 14). For the insertion, if $n_b + n_g = 0$, then there is no deletion to compensate, and the insertion is processed by Reservoir Sampling [Vitter 1985]. That is, if the sample size has not reached its upper bound

⁶Using RP increases the probability that each triangle addition or deletion is observed and thus further decreases the variances of estimates. The variances are roughly inversely proportional to the probability, as stated in Theorem 2.

ALGORITHM 1: Random Pairing: Sampling method that THINKD uses**Inputs:** a dataset: $\mathcal{R}$, a sample: $\mathcal{S}$, an item to be added or deleted: a **Outputs:** a sample after addition or deletion: $\mathcal{S}$

```

1 Procedure INITIALIZE():
2    $\mathcal{S} \leftarrow \emptyset, R \leftarrow \emptyset, n_b \leftarrow 0, n_g \leftarrow 0$ 
3 Procedure INSERT( $a$ ):
4   if  $n_b + n_g = 0$  then
5     if  $|\mathcal{S}| < k$  then  $\mathcal{S} \leftarrow \mathcal{S} \cup \{a\}$ 
6     else if a random number in Bernoulli( $k/|R|$ ) is 1 then
7       replace a random item in  $\mathcal{S}$  with  $a$ 
8   else if a random number in Bernoulli( $n_b/(n_b + n_g)$ ) is 1 then
9      $\mathcal{S} \leftarrow \mathcal{S} \cup \{a\}, n_b \leftarrow n_b + 1$ 
10  else  $n_g \leftarrow n_g + 1$ 
11 Procedure DELETE( $a$ ):
12  if  $a \in \mathcal{S}$  then
13     $\mathcal{S} \leftarrow \mathcal{S} \setminus \{a\}, n_b \leftarrow n_b - 1$ 
14  else  $n_g \leftarrow n_g - 1$ 

```

(i.e., $|\mathcal{S}| < k$), RP adds the inserted item to $\mathcal{S}$ (line 5); otherwise, RP replaces a random item in $\mathcal{S}$ with the inserted item by a certain probability (lines 6–7). If $n_b + n_g > 0$, then RP tosses a coin with probability $n_b/(n_b + n_g)$ (line 8). If the head comes up with probability $n_b/(n_b + n_g)$, RP adds the inserted item to the sample and decrements n_b (line 9). If the tail comes up with probability $n_g/(n_b + n_g)$, RP decrements n_g (line 10).

In high level, RP pairs the inserted item with an uncompensated deletion if possible, and adds the inserted item to the sample if and only if the paired item was in the sample at the time of its deletion. Suppose we have $n_b + n_g > 0$ so that there are uncompensated deletions. Then for the insertion, RP pairs the inserted item with a randomly selected uncompensated deletion, called the “partner” deletion. RP adds the inserted item to the sample if and only if the partner was in the sample at the time of its deletion so that the deletion was “bad”. Since the partner is chosen among n_b “bad” deletions and n_g “good” deletions, the “bad” deletion is chosen as the partner with probability $n_b/(n_b + n_g)$. To implement this, rather than tracing each partner, only the probability $n_b/(n_b + n_g)$ needs to be traced. Hence, maintaining the counters n_b and n_g suffices.

The uniformity of RP and the distribution of the sample size can be theoretically computed. For these, we let $\mathcal{R}^{(t)}$ be the dataset and $\mathcal{S}^{(t)}$ be the sample at time t . Similarly, let $n_b^{(t)}$ and $n_g^{(t)}$ be n_b and n_g at time t . Also let $y^{(t)} = \min(k, |\mathcal{R}^{(t)}| + n_b^{(t)} + n_g^{(t)})$.

The weak uniformity of RP in Lemma 1 is direct from Theorem 1 in [Gemulla et al. 2008], and we show the strong uniformity of RP in Lemma 2.

LEMMA 1 (WEAK UNIFORMITY OF RANDOM PAIRING). *At each fixed time t , all equal-sized subsets of the dataset have the same probability to be the set of samples maintained in Algorithm 1. Formally,*

$$\Pr[\mathcal{S}^{(t)} = \mathcal{A}] = \Pr[\mathcal{S}^{(t)} = \mathcal{B}], \forall t \geq 1, \forall \mathcal{A} \neq \mathcal{B} \subset \mathcal{R}^{(t)} \text{ s.t. } |\mathcal{A}| = |\mathcal{B}|. \quad (2)$$

LEMMA 2 (STRONG UNIFORMITY IN RANDOM PAIRING). *At each fixed time, all equal-sized subsets of the dataset have the same probability to be a subset of the samples maintained in Algorithm 1. Formally,*

$$\Pr[\mathcal{A} \subset \mathcal{S}^{(t)}] = \Pr[\mathcal{B} \subset \mathcal{S}^{(t)}], \forall t \geq 1, \forall \mathcal{A} \neq \mathcal{B} \subset \mathcal{R}^{(t)} \text{ s.t. } |\mathcal{A}| = |\mathcal{B}|. \quad (3)$$

ALGORITHM 2: THINKD_{FAST}: Simple and Fast Version of THINKD**Inputs:** fully dynamic graph stream: $(e^{(1)}, e^{(2)}, \dots)$, sampling probability: $\mathcal{R}$ **Outputs:** estimate of the global triangle count: $\bar{c}$
estimates of the local triangle counts: $c[u]$ for each node u

```

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

```

PROOF. Let $e_i^{\mathcal{A}}$ be the family of size- i subsets of $\mathcal{R}^{(t)}$ including $\mathcal{A}$, and let $e_i^{\mathcal{B}}$ be the family of size- i subsets of $\mathcal{R}^{(t)}$ including $\mathcal{B}$. Then, Equation (3) is obtained as follows:

$$\begin{aligned}
 \Pr[\mathcal{A} \subset \mathcal{S}^{(t)}] &= \sum_i \sum_{C \in e_i^{\mathcal{A}}} \Pr[C = \mathcal{S}^{(t)}] \\
 &= \sum_i \sum_{C \in e_i^{\mathcal{B}}} \Pr[C = \mathcal{S}^{(t)}] = \Pr[\mathcal{B} \subset \mathcal{S}^{(t)}],
 \end{aligned}$$

where the second equality is from Equation (2) in Lemma 1 and $|e_i^{\mathcal{A}}| = |e_i^{\mathcal{B}}|$. $\square$

Moreover, the boundedness, the expectation, and the variance of the sample size of RP in Lemma 3 directly follow from Theorem 2 in [Gemulla et al. 2008].

LEMMA 3 (BOUNDEDNESS, EXPECTATION, AND VARIANCE OF THE SAMPLE SIZE OF RANDOM PAIRING). *At each fixed time t in Algorithm 1, the sample size always satisfies*

$$0 \leq |\mathcal{S}^{(t)}| \leq k, \forall t \geq 1.$$

Also, the expected value and the variance of the sample size are as follows:

$$\mathbb{E}[|\mathcal{S}^{(t)}|] = \frac{|\mathcal{R}^{(t)}| \cdot y^{(t)}}{|\mathcal{R}^{(t)}| + n_b^{(t)} + n_g^{(t)}}, \forall t \geq 1. \quad (4)$$

$$\text{Var}[|\mathcal{S}^{(t)}|] = \frac{(n_b^{(t)} + n_g^{(t)}) \cdot y^{(t)} \cdot (|\mathcal{R}^{(t)}| + n_b^{(t)} + n_g^{(t)} - y^{(t)}) \cdot |\mathcal{R}^{(t)}|}{(|\mathcal{R}^{(t)}| + n_b^{(t)} + n_g^{(t)})^2 \cdot (|\mathcal{R}^{(t)}| + n_b^{(t)} + n_g^{(t)} - 1)}, \forall t \geq 1. \quad (5)$$

4.2 Simple and Fast Version of THINKD: THINKD_{FAST}

THINKD_{FAST}, which is a simple and fast version of THINKD, is described in Algorithm 2. THINKD_{FAST} initially has no sampled edges (line 1). Whenever each element $(\{u, v\}, \delta)$ of the input stream arrives (line 2), THINKD_{FAST} first updates its estimates by calling the procedure UPDATE (line 3). Then, if the element is an addition (i.e., $\delta = +$), THINKD_{FAST} samples the edge $\{u, v\}$ with a given sampling probability $\mathcal{R}$ (line 12) by calling the procedure INSERT (line 4). If the element is a deletion (i.e., $\delta = -$), THINKD_{FAST} removes the edge $\{u, v\}$ from the existing samples (line 14) by calling the procedure DELETE (line 5).

In the procedure UPDATE, THINKD_{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_{FAST} increases the estimates of the global count and the corresponding local counts (line 9). In the case of deletions (i.e., $\delta = -$), since such triangles are those removed from the input stream, THINKD_{FAST} decreases the estimates of the global count and the corresponding local counts (line 10).

As explained in detail in Section 4.4, for unbiased estimates, the amount of change per triangle discovered in lines 9 and 10 should be the reciprocal of the probability that each added or deleted triangle is discovered. This is because this makes the expected amount of changes in the corresponding estimates for each triangle by exactly one. In THINKD_{FAST}, each such triangle $\{u, v, w\}$ is discovered if and only if $\{w, u\}$ and $\{v, w\}$ are in $\mathcal{S}$, and thus the probability is r^2 , as formalized in Lemma 5, which is based on Lemma 4. In both lemmas, we let $X^{(t)}$ be the random number in $Bernoulli(r)$ drawn in line 12 while the t -th element $e^{(t)}$ is processed, and we let $\mathcal{S}^{(t)}$ be $\mathcal{S}$ after the t -th element $e^{(t)}$ is processed. For each edge $\{u, v\}$, we let $l_{uv}^{(t)}$ be the last time that $\{u, v\}$ is added to or removed from $\mathcal{G}$ at time t or earlier. That is,

$$l_{uv}^{(t)} := \max(\{1 \leq s \leq t : e^{(s)} = (\{u, v\}, +) \text{ or } e^{(s)} = (\{u, v\}, -)\}). \quad (6)$$

LEMMA 4. In Algorithm 2, for each time $t \geq 1$ and any edge $\{u, v\} \in \mathcal{E}^{(t)}$, $\{u, v\} \in \mathcal{S}^{(t)}$ if and only if $X^{(l_{uv}^{(t)})} = 1$. That is,

$$\{u, v\} \in \mathcal{S}^{(t)} \iff X^{(l_{uv}^{(t)})} = 1, \forall t \geq 1, \forall \{u, v\} \in \mathcal{E}^{(t)} \quad (7)$$

PROOF. Note that $\{u, v\} \in \mathcal{E}^{(t)}$ implies that $e^{(l_{uv}^{(t)})} = (\{u, v\}, +)$, i.e., the edge $\{u, v\}$ is added at time $l_{uv}^{(t)}$. Then $\{u, v\} \notin \mathcal{E}^{(l_{uv}^{(t)}-1)}$, and since $\mathcal{S}^{(s)} \subset \mathcal{E}^{(s)}$ for all $s \geq 1$, $\{u, v\} \notin \mathcal{S}^{(l_{uv}^{(t)}-1)}$ as well. Therefore,

$$\{u, v\} \in \mathcal{S}^{(l_{uv}^{(t)})} \iff X^{(l_{uv}^{(t)})} = 1. \quad (8)$$

Also, from Equation (6), $e^{(s)} \neq (\{u, v\}, \delta)$ if $l_{uv}^{(t)} < s \leq t$, and hence $\{u, v\}$ is not added after time $l_{uv}^{(t)}$ in Algorithm 1. Hence for all $s \in [l_{uv}^{(t)}, t]$,

$$\{u, v\} \in \mathcal{S}^{(s)} \iff \{u, v\} \in \mathcal{S}^{(l_{uv}^{(t)})}. \quad (9)$$

Combining Equation (8) and Equation (9) with $s = t$ gives Equation (7). $\square$

LEMMA 5 (DISCOVERY PROBABILITY OF TRIANGLES IN THINKD_{FAST}). In THINKD_{FAST}, any two distinct edges in graph $\mathcal{G}^{(t)} = (\mathcal{V}^{(t)}, \mathcal{E}^{(t)})$ are sampled with probability r^2 . That is,

$$Pr[\{u, v\} \in \mathcal{S}^{(t)} \cap \{w, x\} \in \mathcal{S}^{(t)}] = r^2, \forall t \geq 1, \forall \{u, v\} \neq \{w, x\} \in \mathcal{E}^{(t)}. \quad (10)$$

PROOF. Applying Lemma 4 to the edges $\{u, v\}$ and $\{w, x\}$ gives

$$\{u, v\} \in \mathcal{S}^{(t)} \iff X^{(l_{uv}^{(t)})} = 1 \quad \text{and} \quad \{w, x\} \in \mathcal{S}^{(t)} \iff X^{(l_{wx}^{(t)})} = 1. \quad (11)$$

Then, since $X^{(s)}$'s are independent *Bernoulli*(r) and $l_{uv}^{(t)} \neq l_{wx}^{(t)}$, applying Equation (11) with independence of $X^{(l_{uv}^{(t)})}$ and $X^{(l_{wx}^{(t)})}$ gives

$$\begin{aligned} \Pr \left[\{u, v\} \in \mathcal{S}^{(t)} \cap \{w, x\} \in \mathcal{S}^{(t)} \right] &= \Pr \left[X^{(l_{uv}^{(t)})} = 1 \cap X^{(l_{wx}^{(t)})} = 1 \right] \\ &= \Pr \left[X^{(l_{uv}^{(t)})} = 1 \right] \Pr \left[X^{(l_{wx}^{(t)})} = 1 \right] = r^2. \end{aligned} \quad \square$$

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

4.3 Accurate Version of THINKD: THINKD_{ACC}

THINKD_{ACC}, which is an accurate version of THINKD, is described in Algorithm 3. Unlike THINKD_{FAST}, which may discard edges even when memory is not full, THINKD_{ACC} maintains as many samples as possible within a given memory budget k (≥ 2) to minimize information loss.

To this end, THINKD_{ACC} uses a sampling method called RP, described in detail in Section 4.1. 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. Specifically, 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, as shown in Lemma 2:

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

This uniformity makes the probability that each triangle is discovered simple and computationally inexpensive, as shown below.

Updating the set $\mathcal{S}$ of samples using RP is described in lines 11–24. 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 23 and 24). Roughly speaking, n_b and n_g denote the number of deletions that need to be “compensated” by additions (lines 17–19). If there is no deletion to compensate, RP processes each addition of an edge as in Reservoir Sampling [Vitter 1985]. That is, if memory is not full (i.e., $|\mathcal{S}| < k$), RP adds the new edge to $\mathcal{S}$ (line 14), while otherwise, RP replaces a random edge in $\mathcal{S}$ with the new edge with a certain probability (lines 15–16). See Section 4.1 for the definition of n_b and n_g , the details of RP, and the intuition behind the compensation; and we focus on how to use RP for triangle counting in the rest of this section.

Updating the estimates in THINKD_{ACC} is the same as that in THINKD_{FAST} except for the amount of change per triangle discovered in lines 9 and 10. As in THINKD_{FAST}, for unbiased estimates, the amount should be 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 7, if we let $y = \min(k, |\mathcal{E}| + n_b + n_g)$, then the probability of such an event is

$$p(|\mathcal{E}|, n_b, n_g) := \frac{y}{|\mathcal{E}| + n_b + n_g} \times \frac{y - 1}{|\mathcal{E}| + n_b + n_g - 1}. \quad (12)$$

Lemma 7 is based on Lemma 6, and in both lemmas, we let $\mathcal{E}^{(t)}$ be the set of edges remaining (without being deleted) in the input graph stream after the t -th element is processed. We also let $\mathcal{S}^{(t)} \subset \mathcal{E}^{(t)}$ and $y^{(t)}$ be $\mathcal{S}$ and y after the t -th element is processed.

LEMMA 6 (SAMPLING PROBABILITY OF EACH EDGE). *The probability that each edge is sampled in Algorithm 3 is as follows:*

$$Pr[\{u, v\} \in \mathcal{S}^{(t)}] = \frac{y^{(t)}}{|\mathcal{E}^{(t)}| + n_b^{(t)} + n_g^{(t)}}, \quad \forall t \geq 1, \quad \forall \{u, v\} \in \mathcal{E}^{(t)}. \quad (13)$$

PROOF. Let $\mathbf{1}(\{u, v\} \in \mathcal{S}^{(t)})$ be a random variable which is 1 if $\{u, v\} \in \mathcal{S}^{(t)}$ and 0 otherwise. By definition,

$$|\mathcal{S}^{(t)}| = \sum_{\{u, v\} \in \mathcal{E}^{(t)}} \mathbf{1}(\{u, v\} \in \mathcal{S}^{(t)}). \quad (14)$$

Then, by linearity of expectation and Equation (14),

$$\mathbb{E}[|\mathcal{S}^{(t)}|] = \sum_{\{u, v\} \in \mathcal{E}^{(t)}} \mathbb{E}[\mathbf{1}(\{u, v\} \in \mathcal{S}^{(t)})] = \sum_{\{u, v\} \in \mathcal{E}^{(t)}} Pr[\{u, v\} \in \mathcal{S}^{(t)}]. \quad (15)$$

Then, Equation (13) is obtained as follows:

$$\begin{aligned} Pr[\{u, v\} \in \mathcal{S}^{(t)}] &= \frac{1}{|\mathcal{E}^{(t)}|} \sum_{\{w, x\} \in \mathcal{E}^{(t)}} Pr[\{w, x\} \in \mathcal{S}^{(t)}] \\ &= \frac{\mathbb{E}[|\mathcal{S}^{(t)}|]}{|\mathcal{E}^{(t)}|} = \frac{y^{(t)}}{|\mathcal{E}^{(t)}| + n_b^{(t)} + n_g^{(t)}}, \end{aligned}$$

where the first, second, and last equalities are from Equation (3), Equation (15), and Equation (4) in Lemma 3, respectively. $\square$

LEMMA 7 (DISCOVERY PROBABILITY OF TRIANGLES IN THINKD_{ACC}). *In THINKD_{ACC}, any two distinct edges in graph $\mathcal{G}^{(t)} = (\mathcal{V}^{(t)}, \mathcal{E}^{(t)})$ are sampled with probability as in Equation (12). That is, if we let $p^{(t)}$ and $\mathcal{S}^{(t)}$ be the values of Equation (12) and $\mathcal{S}$, resp., in Algorithm 3 after the t -th element $e^{(t)}$ is processed, then*

$$Pr[\{u, v\} \in \mathcal{S}^{(t)} \cap \{w, x\} \in \mathcal{S}^{(t)}] = p^{(t)}, \quad \forall t \geq 1, \quad \forall \{u, v\} \neq \{w, x\} \in \mathcal{E}^{(t)}. \quad (16)$$

PROOF. Let $\mathbf{1}(\{u, v\} \in \mathcal{S}^{(t)})$ be a random variable which is 1 if $\{u, v\} \in \mathcal{S}^{(t)}$ and 0 otherwise. For calculating $Pr[\{u, v\} \in \mathcal{S}^{(t)} \cap \{w, x\} \in \mathcal{S}^{(t)}]$, we expand the covariance sum $\sum_{\{u, v\} \neq \{w, x\}} Cov(\mathbf{1}(\{u, v\} \in \mathcal{S}^{(t)}), \mathbf{1}(\{w, x\} \in \mathcal{S}^{(t)}))$ in two ways and compare them.

First, we use the expansion of the variance of $\mathcal{S}^{(t)}$. From Equation (14),

$$\begin{aligned} Var[|\mathcal{S}^{(t)}|] &= \sum_{\{u, v\} \in \mathcal{E}^{(t)}} Var[\mathbf{1}(\{u, v\} \in \mathcal{S}^{(t)})] \\ &\quad + \sum_{\{u, v\} \neq \{w, x\}} Cov(\mathbf{1}(\{u, v\} \in \mathcal{S}^{(t)}), \mathbf{1}(\{w, x\} \in \mathcal{S}^{(t)})), \end{aligned}$$

and hence the covariance sum can be expanded as

$$\begin{aligned} &\sum_{\{u, v\} \neq \{w, x\}} Cov(\mathbf{1}(\{u, v\} \in \mathcal{S}^{(t)}), \mathbf{1}(\{w, x\} \in \mathcal{S}^{(t)})) \\ &= Var[|\mathcal{S}^{(t)}|] - \sum_{\{u, v\} \in \mathcal{E}^{(t)}} Var[\mathbf{1}(\{u, v\} \in \mathcal{S}^{(t)})]. \end{aligned} \quad (17)$$

For the second term of Equation (17), $Var[x] = \mathbb{E}[x^2] - (\mathbb{E}[x])^2$ implies

$$Var[\mathbf{1}(\{u, v\} \in \mathcal{S}^{(t)})] = Pr[\{u, v\} \in \mathcal{S}^{(t)}] - Pr[\{u, v\} \in \mathcal{S}^{(t)}]^2. \quad (18)$$

ALGORITHM 3: THINKD_{ACC}: Accurate Version of THINKD

Inputs: fully dynamic graph stream: $(e^{(1)}, e^{(2)}, \dots)$, 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

```

1  $S \leftarrow \emptyset, |\mathcal{E}| \leftarrow 0, n_b \leftarrow 0, n_g \leftarrow 0$ 
2 for each element  $e^{(t)} = (\{u, v\}, \delta)$  in the input stream do
3    $\text{UPDATE}(\{u, v\}, \delta)$ 
4   if  $\delta = +$  then  $\text{INSERT}(\{u, v\})$ 
5   else if  $\delta = -$  then  $\text{DELETE}(\{u, v\})$ 
6 Procedure  $\text{UPDATE}(\{u, v\}, \delta)$ :  $\triangleright$  update global and local triangle counts
7   compute  $\hat{N}[u] \cap \hat{N}[v]$   $\triangleright \hat{N}[u]$  and  $\hat{N}[v]$  are obtained from current  $S$ 
8   for each common neighbor  $w \in \hat{N}[u] \cap \hat{N}[v]$  do
9     if  $\delta = +$  then increase  $\bar{c}$ ,  $c[u]$ ,  $c[v]$ , and  $c[w]$  by  $1/p(|\mathcal{E}|, n_b, n_g)$ 
10    else if  $\delta = -$  then decrease  $\bar{c}$ ,  $c[u]$ ,  $c[v]$ , and  $c[w]$  by  $1/p(|\mathcal{E}|, n_b, n_g)$ 
11 Procedure  $\text{INSERT}(\{u, v\})$ :
12    $|\mathcal{E}| \leftarrow |\mathcal{E}| + 1$ 
13   if  $n_b + n_g = 0$  then
14     if  $|S| < k$  then  $S \leftarrow S \cup \{\{u, v\}\}$ 
15     else if a random number in  $\text{Bernoulli}(k/|\mathcal{E}|)$  is 1 then
16       replace a random edge in  $S$  with  $\{u, v\}$ 
17   else if a random number in  $\text{Bernoulli}(n_b/(n_b + n_g))$  is 1 then
18      $S \leftarrow S \cup \{\{u, v\}\}, n_b \leftarrow n_b + 1$ 
19   else  $n_g \leftarrow n_g + 1$ 
20 Procedure  $\text{DELETE}(\{u, v\})$ :
21    $|\mathcal{E}| \leftarrow |\mathcal{E}| - 1$ 
22   if  $\{u, v\} \in S$  then
23      $S \leftarrow S \setminus \{\{u, v\}\}, n_b \leftarrow n_b - 1$ 
24   else  $n_g \leftarrow n_g - 1$ 

```

Hence applying Equation (13) and Equation (18) to Equation (17) gives the covariance sum as

$$\begin{aligned}
& \sum_{\{u, v\} \neq \{w, x\}} \text{Cov}(\mathbf{1}(\{u, v\} \in \mathcal{S}^{(t)}), \mathbf{1}(\{w, x\} \in \mathcal{S}^{(t)})) \\
&= \text{Var}[|\mathcal{S}^{(t)}|] - \sum_{\{u, v\} \in \mathcal{E}^{(t)}} \text{Var}[\mathbf{1}(\{u, v\} \in \mathcal{S}^{(t)})] \\
&= \text{Var}[|\mathcal{S}^{(t)}|] - \sum_{\{u, v\} \in \mathcal{E}^{(t)}} \left(\text{Pr}[\{u, v\} \in \mathcal{S}^{(t)}] - \text{Pr}[\{u, v\} \in \mathcal{S}^{(t)}]^2 \right) \\
&= \text{Var}[|\mathcal{S}^{(t)}|] - |\mathcal{E}^{(t)}| \cdot \frac{y^{(t)} \cdot (|\mathcal{E}^{(t)}| + n_b^{(t)} + n_g^{(t)} - y^{(t)})}{(|\mathcal{E}^{(t)}| + n_b^{(t)} + n_g^{(t)})^2}. \tag{19}
\end{aligned}$$

Second, we directly expand the covariance sum. Expanding the covariance sum with $\text{Cov}(x, y) = \mathbb{E}[xy] - \mathbb{E}[x] \cdot \mathbb{E}[y]$ and applying Equation (13) give

$$\begin{aligned}
& \sum_{\{u,v\} \neq \{w,x\}} \text{Cov}(\mathbf{1}(\{u,v\} \in \mathcal{S}^{(t)}), \mathbf{1}(\{w,x\} \in \mathcal{S}^{(t)})) \\
&= \sum_{\{u,v\} \neq \{w,x\}} \left(\Pr[\{u,v\} \in \mathcal{S}^{(t)} \cap \{w,x\} \in \mathcal{S}^{(t)}] - \Pr[\{u,v\} \in \mathcal{S}^{(t)}] \cdot \Pr[\{w,x\} \in \mathcal{S}^{(t)}] \right) \\
&= \sum_{\{u,v\} \neq \{w,x\}} \left(\Pr[\{u,v\} \in \mathcal{S}^{(t)} \cap \{w,x\} \in \mathcal{S}^{(t)}] - \left(\frac{y^{(t)}}{|\mathcal{E}^{(t)}| + n_b^{(t)} + n_g^{(t)}} \right)^2 \right) \\
&= \sum_{\{u,v\} \neq \{w,x\}} \left(\Pr[\{u,v\} \in \mathcal{S}^{(t)} \cap \{w,x\} \in \mathcal{S}^{(t)}] \right) - \frac{y^{(t)} \cdot y^{(t)} \cdot |\mathcal{E}^{(t)}| \cdot (|\mathcal{E}^{(t)}| - 1)}{(|\mathcal{E}^{(t)}| + n_b^{(t)} + n_g^{(t)})^2}. \tag{20}
\end{aligned}$$

Now, the probability sum $\sum_{\{u,v\} \neq \{w,x\}} \Pr[\{u,v\} \in \mathcal{S}^{(t)} \cap \{w,x\} \in \mathcal{S}^{(t)}]$ can be obtained by comparing two expansions Equation (19) and Equation (20) of the covariance sum and applying Equation (5) in Lemma 3 as

$$\begin{aligned}
& \sum_{\{u,v\} \neq \{w,x\}} \Pr[\{u,v\} \in \mathcal{S}^{(t)} \cap \{w,x\} \in \mathcal{S}^{(t)}] \\
&= \sum_{\{u,v\} \neq \{w,x\}} \text{Cov}(\mathbf{1}(\{u,v\} \in \mathcal{S}^{(t)}), \mathbf{1}(\{w,x\} \in \mathcal{S}^{(t)})) + \frac{y^{(t)} \cdot y^{(t)} \cdot |\mathcal{E}^{(t)}| \cdot (|\mathcal{E}^{(t)}| - 1)}{(|\mathcal{E}^{(t)}| + n_b^{(t)} + n_g^{(t)})^2} \\
&= \text{Var}[|\mathcal{S}^{(t)}|] - |\mathcal{E}^{(t)}| \cdot \frac{y^{(t)} \cdot (|\mathcal{E}^{(t)}| + n_b^{(t)} + n_g^{(t)} - y^{(t)})}{(|\mathcal{E}^{(t)}| + n_b^{(t)} + n_g^{(t)})^2} + \frac{y^{(t)} \cdot y^{(t)} \cdot |\mathcal{E}^{(t)}| \cdot (|\mathcal{E}^{(t)}| - 1)}{(|\mathcal{E}^{(t)}| + n_b^{(t)} + n_g^{(t)})^2} \\
&= \frac{y^{(t)} \cdot (y^{(t)} - 1) \cdot |\mathcal{E}^{(t)}| \cdot (|\mathcal{E}^{(t)}| - 1)}{(|\mathcal{E}^{(t)}| + n_b^{(t)} + n_g^{(t)}) \cdot (|\mathcal{E}^{(t)}| + n_b^{(t)} + n_g^{(t)} - 1)}. \tag{21}
\end{aligned}$$

Then, Equation (16) is obtained by Equation (21) as follows:

$$\begin{aligned}
& \Pr[\{u,v\} \in \mathcal{S}^{(t)} \cap \{w,x\} \in \mathcal{S}^{(t)}] \\
&= \frac{1}{|\mathcal{E}^{(t)}| \cdot (|\mathcal{E}^{(t)}| - 1)} \left(\sum_{\{u,v\} \neq \{w,x\}} \Pr[\{u,v\} \in \mathcal{S}^{(t)} \cap \{w,x\} \in \mathcal{S}^{(t)}] \right) \\
&= \frac{1}{|\mathcal{E}^{(t)}| \cdot (|\mathcal{E}^{(t)}| - 1)} \cdot \frac{y^{(t)} \cdot (y^{(t)} - 1) \cdot |\mathcal{E}^{(t)}| \cdot (|\mathcal{E}^{(t)}| - 1)}{(|\mathcal{E}^{(t)}| + n_b^{(t)} + n_g^{(t)}) \cdot (|\mathcal{E}^{(t)}| + n_b^{(t)} + n_g^{(t)} - 1)} \\
&= \frac{y^{(t)}}{|\mathcal{E}^{(t)}| + n_b^{(t)} + n_g^{(t)}} \cdot \frac{y^{(t)} - 1}{|\mathcal{E}^{(t)}| + n_b^{(t)} + n_g^{(t)} - 1} = p^{(t)},
\end{aligned}$$

where the first equality is from Equation (3). $\square$

(Dis)advantages of THINKD_{ACC}: Within the same memory budget, THINKD_{ACC} is slower than THINKD_{FAST} since THINKD_{ACC} maintains and processes more samples on average. However, THINKD_{ACC} is more accurate than THINKD_{FAST} by utilizing more samples. These are shown empirically in Section 5.3 and Section 5.4.

Reducing estimation errors by sacrificing unbiasedness: The estimates (i.e., $\bar{c}$ and $c[u]$ for each node u) in Algorithms 2 and 3 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.

4.4 Accuracy Analyses

We prove that $\text{THINKD}_{\text{FAST}}$ and $\text{THINKD}_{\text{ACC}}$ maintain unbiased estimates, whose expected values are equal to the true global and local triangle counts. Then, we analyze the variances of the estimates that $\text{THINKD}_{\text{FAST}}$ maintains. To this end, for each variable (e.g., $\bar{c}$) in Algorithms 2 and 3, 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.⁷

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.²

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 8 formalizes the relationship between these concepts and the number of triangles.

LEMMA 8 (COUNT OF TRIANGLES IN THE CURRENT GRAPH). *The count of triangles in the current graph equals the count of added triangles subtracted by the count of deleted triangles. Formally,*

$$|\mathcal{T}^{(t)}| = |\mathcal{A}^{(t)}| - |\mathcal{D}^{(t)}|, \quad \forall t \geq 1, \quad (22)$$

$$|\mathcal{T}^{(t)}[u]| = |\mathcal{A}^{(t)}[u]| - |\mathcal{D}^{(t)}[u]|, \quad \forall t \geq 1, \quad \forall u \in \mathcal{V}^{(t)}. \quad (23)$$

PROOF. When $t = 1$, then $\mathcal{T}^{(1)} = \mathcal{A}^{(1)} = \mathcal{D}^{(1)} = \emptyset$ holds, and hence Equation (22) trivially holds. Hence we assume that $t \geq 2$ from now on. First, we show that for each time $s \geq 2$,

$$|\mathcal{T}^{(s)}| - |\mathcal{T}^{(s-1)}| = |\mathcal{A}^{(s)} \setminus \mathcal{A}^{(s-1)}| - |\mathcal{D}^{(s)} \setminus \mathcal{D}^{(s-1)}|. \quad (24)$$

To show this, we show the following relations,

$$|\mathcal{A}^{(s)} \setminus \mathcal{A}^{(s-1)}| = |\mathcal{T}^{(s)} \setminus \mathcal{T}^{(s-1)}|, \quad (25)$$

$$|\mathcal{D}^{(s)} \setminus \mathcal{D}^{(s-1)}| = |\mathcal{T}^{(s-1)} \setminus \mathcal{T}^{(s)}|. \quad (26)$$

For Equation (25), note that

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

and

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

hold, and hence Equation (25) holds. Similarly,

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

⁷Note 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.

and

$$\mathcal{T}^{(s-1)} \setminus \mathcal{T}^{(s)} = \left\{ \{u, v, w\} : \{u, v, w\} \in \mathcal{T}^{(s-1)} \text{ and } \{u, v, w\} \notin \mathcal{T}^{(s)} \right\}$$

hold, and hence Equation (26) holds. Then, Equation (25) and Equation (26) imply

$$\begin{aligned} |\mathcal{A}^{(s)} \setminus \mathcal{A}^{(s-1)}| + |\mathcal{T}^{(s-1)}| &= |\mathcal{T}^{(s)} \setminus \mathcal{T}^{(s-1)}| + |\mathcal{T}^{(s-1)}| = |\mathcal{T}^{(s-1)} \cup \mathcal{T}^{(s)}| \\ &= |\mathcal{T}^{(s-1)} \setminus \mathcal{T}^{(s)}| + |\mathcal{T}^{(s)}| = |\mathcal{D}^{(s)} \setminus \mathcal{D}^{(s-1)}| + |\mathcal{T}^{(s)}|, \end{aligned}$$

and hence Equation (24) holds. Then, summing up Equation (24) from $s = 2$ to t yields

$$|\mathcal{T}^{(t)}| - |\mathcal{T}^{(1)}| = \sum_{s=2}^t |\mathcal{A}^{(s)} \setminus \mathcal{A}^{(s-1)}| - \sum_{s=2}^t |\mathcal{D}^{(s)} \setminus \mathcal{D}^{(s-1)}|. \quad (27)$$

Then, $\{\mathcal{A}^{(s)} \setminus \mathcal{A}^{(s-1)}\}_{s=2}^t$ being disjoint over s implies

$$\sum_{s=2}^t |\mathcal{A}^{(s)} \setminus \mathcal{A}^{(s-1)}| = \left| \bigcup_{s=2}^t (\mathcal{A}^{(s)} \setminus \mathcal{A}^{(s-1)}) \right| = |\mathcal{A}^{(t)} \setminus \mathcal{A}^{(1)}|, \quad (28)$$

and similarly,

$$\sum_{s=2}^t |\mathcal{D}^{(s)} \setminus \mathcal{D}^{(s-1)}| = |\mathcal{D}^{(t)} \setminus \mathcal{D}^{(1)}|. \quad (29)$$

holds. Then, applying Equation (28), Equation (29), and $\mathcal{T}^{(1)} = \mathcal{A}^{(1)} = \mathcal{D}^{(1)} = \emptyset$ to Equation (27) yields that for all $t \geq 2$,

$$|\mathcal{T}^{(t)}| = |\mathcal{A}^{(t)}| - |\mathcal{D}^{(t)}|,$$

which completes the proof of Equation (22).

For Equation (23), replacing $\mathcal{T}^{(s)}$ by $\mathcal{T}^{(s)}[u]$, $\mathcal{A}^{(s)}$ by $\mathcal{A}^{(s)}[u]$, and $\mathcal{D}^{(s)}$ by $\mathcal{D}^{(s)}[u]$ and repeating above give a proof. $\square$

Based on these concepts, we prove that $\text{THINKD}_{\text{FAST}}$ and $\text{THINKD}_{\text{ACC}}$ 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 . This follows from the fact that the amount of change in estimates for each observed triangle addition or deletion is the reciprocal of the probability that each triangle addition or deletion is observed. Then, by Lemma 8, the expected value of $\bar{c}$ equals 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.

THEOREM 1 (“ANY TIME” UNBIASEDNESS OF THINKD). *THINKD gives unbiased estimates at any time. Formally, in Algorithms 2 and 3,*

$$\mathbb{E}[\bar{c}^{(t)}] = |\mathcal{T}^{(t)}|, \quad \forall t \geq 1, \quad (30)$$

$$\mathbb{E}[c^{(t)}[u]] = |\mathcal{T}^{(t)}[u]|, \quad \forall t \geq 1, \quad \forall u \in \mathcal{V}^{(t)}. \quad (31)$$

PROOF. Consider a triangle $(\{u, v, w\}, s) \in \mathcal{A}^{(t)}$, and let $e^{(s)} = (\{u, v\}, +)$ without loss of generality. The amount $\alpha_{uvw}^{(s)}$ 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 2 or Algorithm 3 is

$$\alpha_{uvw}^{(s)} = \begin{cases} 1/r^2 & \text{if } \{v, w\} \in \mathcal{S}^{(s-1)} \text{ and } \{w, u\} \in \mathcal{S}^{(s-1)} \text{ in Algorithm 2} \\ 1/p^{(s-1)} & \text{if } \{v, w\} \in \mathcal{S}^{(s-1)} \text{ and } \{w, u\} \in \mathcal{S}^{(s-1)} \text{ in Algorithm 3} \\ 0 & \text{otherwise.} \end{cases}$$

Then, from Equation (10) and Equation (16), the following equation holds:

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

Hence,

$$\mathbb{E}[\alpha_{uvw}^{(s)}] = 1. \quad (32)$$

Consider a triangle $(\{u, v, w\}, s) \in \mathcal{D}^{(t)}$, and let $e^{(s)} = (\{u, v\}, -)$ without loss of generality. The amount $\beta_{uvw}^{(s)}$ 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 10 of Algorithm 2 or Algorithm 3 is

$$\beta_{uvw}^{(s)} = \begin{cases} -1/r^2 & \text{if } \{v, w\} \in \mathcal{S}^{(s-1)} \text{ and } \{w, u\} \in \mathcal{S}^{(s-1)} \text{ in Algorithm 2} \\ -1/p^{(s-1)} & \text{if } \{v, w\} \in \mathcal{S}^{(s-1)} \text{ and } \{w, u\} \in \mathcal{S}^{(s-1)} \text{ in Algorithm 3} \\ 0 & \text{otherwise.} \end{cases}$$

Then, from Equation (10) and Equation (16), the following equation holds:

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

Hence,

$$\mathbb{E}[\beta_{uvw}^{(s)}] = -1. \quad (33)$$

By definition, the following holds:

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

By linearity of expectation, Equation (32), Equation (33), and Lemma 8, the following holds:

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

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

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

By linearity of expectation, Equation (32), Equation (33), and Lemma 8, the following holds:

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

In Appendix B, we prove the formulas for the variances of estimates given by THINKD_{FAST}. Theorem 2 is implied by the formulas. Note that r^2 in Equation (34) and Equation (35) is the discovery probability shown in Lemma 5.

THEOREM 2 (VARIANCE OF THINKD_{FAST}). *Given an input graph stream, the variances of estimates maintained by THINKD_{FAST} with the sampling probability $\mathcal{R}$ is proportional to $1/r^2$. Formally, in Algorithm 2,*

$$\text{Var}[\bar{c}^{(t)}] = O(1/r^2), \quad \forall t \geq 1, \quad (34)$$

$$\text{Var}[c^{(t)}[u]] = O(1/r^2), \forall t \geq 1, \forall u \in \mathcal{V}^{(t)}. \quad (35)$$

PROOF. See Theorem 5 in Appendix B. $\square$

4.5 Complexity Analyses

We analyze the time and space complexities of $\text{THINKD}_{\text{FAST}}$ and $\text{THINKD}_{\text{ACC}}$. In our analyses, we use $\mathcal{V}^{(t)} := \bigcup_{s=1}^t \mathcal{V}^{(s)}$ 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, $\text{THINKD}_{\text{FAST}}$ and $\text{THINKD}_{\text{ACC}}$ maintain one estimate for the global triangle count and at most $|\mathcal{V}^{(t)}|$ estimates for the local triangle counts. In addition, $\text{THINKD}_{\text{FAST}}$ maintains $|\mathcal{E}^{(t)}| \cdot r$ edges on average, while $\text{THINKD}_{\text{ACC}}$ maintains up to k edges. Thus, the average space complexities of $\text{THINKD}_{\text{FAST}}$ and $\text{THINKD}_{\text{ACC}}$ are $O(|\mathcal{E}^{(t)}| \cdot r + |\mathcal{V}^{(t)}|)$ and $O(k + |\mathcal{V}^{(t)}|)$, 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 $\text{THINKD}_{\text{FAST}}$ in Theorem 3, which implies Corollary 1, and the worst-case time complexity of $\text{THINKD}_{\text{ACC}}$ in Theorem 4. Corollary 1 and Theorem 4 state that, given a fixed memory budget k , $\text{THINKD}_{\text{FAST}}$ and $\text{THINKD}_{\text{ACC}}$ scale linearly with the number of elements in the input stream.

THEOREM 3 (TIME COMPLEXITY OF $\text{THINKD}_{\text{FAST}}$). *Algorithm 2 takes $O(t + t^2r)$ on average 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 $O(1 + \mathbb{E}[|\hat{\mathcal{N}}[u]| + |\hat{\mathcal{N}}[v]|]) = O(1 + \mathbb{E}[|\mathcal{S}|]) = O(1 + sr)$ on average. Hence, processing the first t elements takes $\sum_{s=1}^t O(1 + sr) = O(t + t^2r)$ on average. $\square$

Collorary 1 (Time Complexity of $\text{THINKD}_{\text{FAST}}$ with Fixed Memory k). If $r = O(k/t)$ for a constant k (≥ 1), then Algorithm 2 takes $O(tk)$ on average to process the first t elements in the input stream.

THEOREM 4 (TIME COMPLEXITY OF $\text{THINKD}_{\text{ACC}}$). *Algorithm 3 takes $O(tk)$ to process the first t elements in the input stream.*

PROOF. In Algorithm 3, 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 $O(tk)$. $\square$

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?
- **Q5. Application to Social Network Analysis:** Is THINKD useful for accurate, space-efficient, and incremental estimation of measures that are widely used in social network analysis and network science?

Additionally, in appendices, our experiments for answering the following questions are reviewed:

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
Friendster [Yang and Leskovec 2015]	65.6M	1.81B	Friendship
Orkut [Mislove et al. 2007]	3.07M	117M	Friendship
Flickr [Mislove et al. 2007]	2.30M	22.8M	Friendship
Patent [Hall et al. 2001]	3.77M	16.5M	Citation
Youtube [Mislove et al. 2007]	3.22M	9.38M	Friendship
BerkStan [Leskovec et al. 2009]	685K	6.65M	Web
Facebook [Viswanath et al. 2009]	63.7K	817K	Friendship
Epinion [Massa and Avesani 2005]	132K	711K	Trust
Random ($\approx 800\text{GB}$)*	1M	0.1B-100B	Synthetic

*We used 4 bytes, 8 bytes, and 1 bit to represent a node, an edge, and a sign in the stream, respectively.

If 20 bits and 40 bits were used to represent a node and an edge, the stream would become about 500 GB.

- **Q6. Speed and Accuracy in Multigraph Streams (Appendix C):** Is a variant of THINKD for triangle counting in multigraph streams (i.e., graph streams with parallel edges) faster and more accurate than its best competitors?
- **Q7. Application to Anomaly Detection (Appendix D):** Does THINKD-SPOT, which is based on THINKD, detect suddenly emerging dense subgraphs faster and more accurately than its best competitors?

5.1 Experimental Settings

Machines: We used a PC with a 3.60 GHz Intel i7-4790 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 randomly shuffle them⁸, (b) choose $\alpha\%$ of the edges and create the deletions of them, and (c) locate each deletion in a random position after the corresponding addition. We set α to 20% unless otherwise stated (see Section 5.5 for its effect on accuracy). The created streams were streamed from the disk.

Implementations: We implemented THINKD_{FAST} (Section 4.2), THINKD_{ACC} (Section 4.3), TRIEST_{FD} [De Stefani et al. 2017], TRIEST_{IMPR} [De Stefani et al. 2017], ESD [Han and Sethu 2017], and MASCOT [Lim et al. 2018] commonly in Java. In all of them, sampled edges are stored in the adjacency list format, and as described in the last paragraph of Section 4.3, estimates are lower bounded by zero unless otherwise stated.

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. Then, we measured the accuracy of each algorithm using the following metrics:

- *Global Error* (the lower the better): $|x - \hat{x}|/x$,
- *RMSE* (the lower the better): $\sqrt{\frac{1}{|\mathcal{V}|} \sum_{u \in \mathcal{V}} (x[u] - \hat{x}[u])^2}$,

⁸Instead, additions can be ordered chronologically when timestamps of edges are given. We obtained consistent experimental results when we ordered additions randomly and chronologically.

Epinion Dataset: **Facebook Dataset:**

Fig. 2. **THINKD is provably accurate and scalable.** (a, e) THINKD is “any time”, maintaining estimates while the input graph evolves. (b–c, f–g) THINKD_{ACC} is the most accurate with the smallest confidence intervals over the entire stream. THINKD_{FAST} becomes more accurate than TRIEST_{FD} as the input graph grows over time. (d, h) THINKD provides unbiased estimates, whose expected values are equal to the true triangle counts, and their variances are small.

— *Rank Correlation* (the higher the better): Spearman’s rank correlation coefficient [Spearman 1904] between $\{(u, x[u])\}_{u \in \mathcal{V}}$ and $\{(u, \hat{x}[u])\}_{u \in \mathcal{V}}$.

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_{FAST}, THINKD_{ACC}, and TRIEST_{FD}, whose parameters were set so that on average 10% of the edges are stored at the end of each graph stream. Figures 2(d) and 2(h) show the distributions of the estimates at the end of the Facebook and Epinion datasets, respectively. The means of the estimates were close to the true triangle count, consistently with Theorem 1 (i.e., unbiasedness of THINKD). Moreover, THINKD_{ACC} and THINKD_{FAST} gave estimates with smaller variances than TRIEST_{FD}. Figures 2(b), 2(c), 2(f), and 2(g) show how the 95% confidence intervals, estimated from 10,000 trials, changed over early and late parts of the datasets. THINKD_{ACC} was consistently the most accurate with the smallest confidence intervals. THINKD_{FAST} became more accurate than TRIEST_{FD} as the input graph grew over time. The same trend can be seen in Figures 2(a) and 2(e), where we show how an estimate from a single trial of each algorithm changed over the entire datasets.

THINKD scales linearly (Corollary 1 and Theorem 4). We measured the elapsed times taken by THINKD_{FAST} and THINKD_{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

Fig. 3. **THINKD is scalable.** THINKD scales linearly with the size of the input stream.

10^7 edges are stored at the end of each input stream. Figure 3(a) shows the results in the Random datasets, which were created by the Erdős-Rényi model. Both THINKD_{FAST} and THINKD_{ACC} scaled linearly with the number of elements, as expected in Corollary 1 and Theorem 4. Notice that the largest dataset contains 100 billion elements. As seen in Figure 3(b) and Figure 3(c), we obtained similar trends in graph streams with realistic structures created by sampling different numbers of elements from the Friendster and Orkut datasets.

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_{FAST}, THINKD_{ACC}, and TRIEST_{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.⁹ As seen in Figure 4, THINKD_{FAST} and THINKD_{ACC} consistently gave the best trade-off between space and accuracy. Specifically, within the same memory budget, THINKD_{ACC} was up to 4× and 4.3× **more accurate** than TRIEST_{FD} in terms of global error and Root Mean Square Error (RMSE), respectively. Between our algorithms, THINKD_{ACC} consistently outperformed THINKD_{FAST} in terms of accuracy.

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 Section 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 Figure 5, THINKD_{FAST} and THINKD_{ACC} consistently gave the best trade-off between speed and accuracy. Specifically, for the same global error and RMSE, THINKD_{FAST} was up to 2.2× **faster** than TRIEST_{FD}. Between our algorithms, THINKD_{FAST} consistently outperformed THINKD_{ACC} in terms of speed.

5.5 Q4. Effects of Deletions (THINKD is Consistently Accurate)

We measured how the ratio of deleted edges (i.e., α in Section 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 Figure 6, all algorithms that support edge deletions became more accurate as input graphs became smaller with more deletions. THINKD_{FAST} and THINKD_{ACC} were similarly accurate with MASCOT and TRIEST_{IMPR}, respectively, in the streams without deletions. In the streams with deletions, which MASCOT and TRIEST_{IMPR} cannot handle,

⁹We used a workstation with 2.67 GHz Intel Xeon E7-8837 CPUs and 1 TB memory for the Friendster dataset.

Fig. 4. **THINKD is accurate.** THINKD provides the best trade-off between space and accuracy. In particular, THINKD_{ACC} is up to 4.3X **more accurate** than TRIEST_{FD} within the same memory budget. Error bars denote ± 1 standard error. ESD is inapplicable to local triangle counting.

Fig. 5. **THINKD is fast.** THINKD provides the best trade-off between speed and accuracy. In particular, THINKD_{FAST} is up to 2.2× faster than TRIEST_{FD} when they are similarly accurate. Error bars denote ± 1 standard error. ESD is inapplicable to local triangle counting.

Fig. 6. **THINKD is consistently accurate regardless of the ratio of deleted edges.** Error bars denote ± 1 standard error. $\text{TRIESTE}_{\text{IMPR}}$ and MASCOT are inapplicable when there are deletions. ESD is inapplicable to local triangle counting.

$\text{THINKD}_{\text{FAST}}$ and $\text{THINKD}_{\text{ACC}}$ were $1.8 - 3.4\times$ more accurate than $\text{TRIEST}_{\text{FD}}$ regardless of the ratio of deleted edges.

5.6 Q5. Application to Social Network Analysis

We show that THINKD can be used for accurate and space-efficient estimation of the following measures, which are widely used in social network analysis and network science [Holland and Leinhardt 1971; Kemper 2009; Luce and Perry 1949; Wasserman and Faust 1994; Watts and Strogatz 1998], in fully-dynamic graph streams:

- **Transitivity Ratio (TR)** [Luce and Perry 1949]: The *transitivity ratio* of a graph $\mathcal{G}^{(t)}$ is defined as

$$\text{TR}(\mathcal{G}^{(t)}) := \frac{6 \times |\mathcal{T}^{(t)}|}{\sum_{v \in \mathcal{V}^{(t)}} (d^{(t)}[v] \times (d^{(t)}[v] - 1))},$$

where $d^{(t)}[v]$ is the degree of $v \in \mathcal{V}^{(t)}$ in $\mathcal{G}^{(t)}$.

- **Local Clustering Coefficients (LCC)** [Watts and Strogatz 1998]: The *local clustering coefficient* of each node $v \in \mathcal{V}^{(t)}$ in a graph $\mathcal{G}^{(t)}$ is defined as

$$\text{LCC}(\mathcal{G}^{(t)}, v) := \frac{2 \times |\mathcal{T}^{(t)}[v]|}{d^{(t)}[v] \times (d^{(t)}[v] - 1)},$$

where $d^{(t)}[v]$ is the degree of $v \in \mathcal{V}^{(t)}$ in $\mathcal{G}^{(t)}$.

- **Global Clustering Coefficient (GCC)** [Watts and Strogatz 1998]: The *global clustering coefficient* of a graph $\mathcal{G}^{(t)}$ is defined as

$$\text{GCC}(\mathcal{G}^{(t)}) := \frac{1}{|\mathcal{V}^{(t)}|} \sum_{v \in \mathcal{V}^{(t)}} \text{LCC}(\mathcal{G}^{(t)}, v).$$

Specifically, to estimate these graph measures, the exact degree of each node $v \in \mathcal{V}^{(t)}$ (i.e., $d^{(t)}[v]$) was computed incrementally while $\bar{c}^{(t)}$ and $c^{(t)}[u]$ for each node $v \in \mathcal{V}^{(t)}$ were computed in $\text{THINKD}_{\text{FAST}}$, $\text{THINKD}_{\text{ACC}}$, and $\text{TRIEST}_{\text{FD}}$. Their parameters were set so that on average 30% of the

Table 4. THINKD is Useful for Social Network Analysis (SNA).

	SNA Measures	Transitivity Ratio	Global Clustering Coefficients	Local Clustering Coefficients	
Datasets	Evaluation Metrics	Global Error*	Global Error*	RMSE*	Rank Correlation**
Facebook	TRIEST _{FD}	0.0051	0.019	0.63	0.41
	THINKD _{FAST}	0.0046	0.010	0.26	0.56
	THINKD _{ACC}	0.0026	0.010	0.15	0.68
Epinion	TRIEST _{FD}	0.0083	0.036	0.74	0.51
	THINKD _{FAST}	0.0064	0.012	0.29	0.63
	THINKD _{ACC}	0.0038	0.014	0.17	0.74
BerkStan	TRIEST _{FD}	0.0083	0.016	1.56	0.35
	THINKD _{FAST}	0.0060	0.011	0.72	0.47
	THINKD _{ACC}	0.0037	0.007	0.38	0.58

THINKD enables accurate and space-efficient estimation of widely-used graph measures in fully-dynamic graph streams.

*the lower the better.

**the higher the better.

edges are stored at the end of each input graph stream.¹⁰ Note that incrementally computing the degrees of all nodes takes $O(t)$ time and requires $O(n)$ space.¹¹ Then, replacing $|\mathcal{T}^{(t)}|$ and $|\mathcal{T}^{(t)}[v]|$ in the above definitions by $c^{(t)}$ and $c^{(t)}[v]$, respectively, we estimated the measures. In all the algorithms, the additional time taken for incrementally computing the degrees of all nodes and estimating graph measures was negligible compared to the time taken for estimating the counts of triangles. To evaluate the accuracies of the estimates obtained by different algorithms, we computed the evaluation metrics defined in Section 5.1 at the end of each input stream. As summarized in Table 4, where each evaluation metric was averaged over 10 trials, THINKD_{ACC} and THINKD_{FAST} estimated all the considered graph measures more accurately than TRIEST_{FD} within the same memory budget.

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. We theoretically and empirically show that THINKD has the following advantages:

- **Accurate:** THINKD is up to $4.3\times$ more accurate than its best competitors within the same memory budget (Figure 4).
- **Fast:** THINKD is up to $2.2\times$ faster than its best competitors with similar accuracies (Figure 5). THINKD processes *terabyte*-scale graph streams with linear scalability (Figure 2, 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.

Additionally, in Appendix D, we apply THINKD to the task of detecting suddenly emerging dense subgraphs and show its advantages over state-of-the-art methods.

¹⁰When estimating global clustering coefficients, we computed $c^{(t)}[u]$ for each node $u \in \mathcal{V}^{(t)}$ without lower bounding it by zero.

¹¹For each addition $(\{u, v\}, +)$, increase $d^{(t)}[u]$ and $d^{(t)}[v]$ by 1, and for each deletion $(\{u, v\}, -)$, decrease $d^{(t)}[u]$ and $d^{(t)}[v]$ by 1.

Reproducibility: The source code and datasets used in the article are available at <http://dmlab.kaist.ac.kr/~kijungs/codes/thinkd/>.

APPENDIX

A TOY EXAMPLE FOR THINKD AND ITS COMPETITORS (TRIEST_{FD} AND ESD)

Figure 7 shows an artificial graph stream that highlights the (dis)advantages of THINKD and its competitors (i.e., TRIEST_{FD} and ESD). First, ESD runs out of memory (O.O.M.) at $t \geq 4$ since it stores all edges of the input graph. This implies that ESD has limited scalability. In the case of TRIEST_{FD}, its estimation at $t = 7$ is inaccurate since it does not update estimates if the current edge (i.e., $\{b, c\}$) is not stored but discarded. On the other hand, THINKD updates its estimates whenever an update (i.e., an insertion or a deletion) arrives. Hence, THINKD_{FAST} and THINKD_{ACC} estimate the global triangle count exactly at $t = 7$. THINKD_{FAST} suffers from information loss at $t = 6$ and 7 since it discards edges (i.e., $\{c, e\}$ and $\{b, c\}$) even when the number of edges in memory is less than the given budget. On the other hand, within the given memory budget, THINKD_{ACC} tries to maintain as many edges as possible in memory. Note that we set the memory budget to 3 for all methods to make a fair comparison.

B VARIANCE ANALYSIS

We let $l_{uv}^{(t)}$ be the last time that edge $\{u, v\}$ is added to or removed from $\mathcal{G}$ at time t or earlier. For each added or deleted triangle $(\{u, v, w\}, s) \in \mathcal{A}^{(t)} \cup \mathcal{D}^{(t)}$, we use $1_{(\{u, v, w\}, s)}$ to denote the time when its first edge has arrived and $2_{(\{u, v, w\}, s)}$ to denote the time when its second edge has arrived. Formally,

$$1_{(\{u, v, w\}, s)} := \min(l_{uv}^{(s)}, l_{vw}^{(s)}, l_{wu}^{(s)}), \quad 2_{(\{u, v, w\}, s)} := \text{median}(l_{uv}^{(s)}, l_{vw}^{(s)}, l_{wu}^{(s)}).$$

Then, we define the type of each triangle pair in Definition 3.

Definition 3 (Types of Triangle Pairs). The type of each ordered pair of two distinct triangles $\tau \neq \omega \in \mathcal{A}^{(t)} \cup \mathcal{D}^{(t)}$ is defined as follows:

Time	Change	True Global Count	ThinkD _{Fast} (r = 0.5)		ThinkD _{Acc} (budget = 3)		TRIEST _{FD} (budget = 3)		ESD (budget = 3)	
			Sampled Edges	Estimate	Sampled Edges	Estimate	Sampled Edges	Estimate	Sampled Edges	Estimate
1	$\{a, b\}, +$	0	$\{a, b\}$	0	$\{a, b\}$	0	$\{a, b\}$	0	$\{a, b\}$	0
2	$\{a, c\}, +$	0	$\{a, b\}, \{a, c\}$	0	$\{a, b\}, \{a, c\}$	0	$\{a, b\}, \{a, c\}$	0	$\{a, b\}, \{a, c\}$	0
3	$\{a, d\}, +$	0	$\{a, b\}, \{a, c\}$	0	$\{a, b\}, \{a, c\}, \{a, d\}$	0	$\{a, b\}, \{a, c\}, \{a, d\}$	0	$\{a, b\}, \{a, c\}, \{a, d\}$	0
4	$\{a, e\}, +$	0	$\{a, b\}, \{a, c\}, \{a, e\}$	0	$\{a, b\}, \{a, c\}, \{a, e\}$	0	$\{a, b\}, \{a, c\}, \{a, e\}$	0	O.O.M.	
5	$\{a, e\}, -$	0	$\{a, b\}, \{a, c\}$	0	$\{a, b\}, \{a, c\}$	0	$\{a, b\}, \{a, c\}$	0	O.O.M.	
6	$\{c, e\}, +$	0	$\{a, b\}, \{a, c\}$	0	$\{a, b\}, \{a, c\}, \{c, e\}$	0	$\{a, b\}, \{a, c\}, \{c, e\}$	0	O.O.M.	
7	$\{b, c\}, +$	1	$\{a, b\}, \{a, c\}$	1	$\{a, b\}, \{a, c\}, \{c, e\}$	1	$\{a, b\}, \{a, c\}, \{c, e\}$	0	O.O.M.	

Fig. 7. A toy example that highlights the (dis)advantages of THINKD and its competitors. ESD runs out of memory (O.O.M.) at $t \geq 4$ since it stores all edges of the input graph. TRIEST_{FD} is inaccurate at $t = 7$ since it does not update estimates if an incoming edge is not stored but discarded. However, THINKD_{FAST} and THINKD_{ACC} produce accurate estimates although THINKD_{FAST} suffers from information loss at $t = 6$ and 7.

$$Type_{(\tau, \omega)} = \begin{cases} 1, & \text{if } \tau \in \mathcal{A}^{(t)} \text{ and } \omega \in \mathcal{A}^{(t)} \text{ and } |\{1_\tau, 2_\tau\} \cap \{1_\omega, 2_\omega\}| = 1, \\ 2, & \text{if } \tau \in \mathcal{D}^{(t)} \text{ and } \omega \in \mathcal{D}^{(t)} \text{ and } |\{1_\tau, 2_\tau\} \cap \{1_\omega, 2_\omega\}| = 1, \\ 3, & \text{if } \tau \in \mathcal{A}^{(t)} \text{ and } \omega \in \mathcal{D}^{(t)} \text{ and } |\{1_\tau, 2_\tau\} \cap \{1_\omega, 2_\omega\}| = 1, \\ 4, & \text{if } \tau \in \mathcal{D}^{(t)} \text{ and } \omega \in \mathcal{A}^{(t)} \text{ and } |\{1_\tau, 2_\tau\} \cap \{1_\omega, 2_\omega\}| = 1, \\ 5, & \text{if } \tau \in \mathcal{A}^{(t)} \text{ and } \omega \in \mathcal{A}^{(t)} \text{ and } |\{1_\tau, 2_\tau\} \cap \{1_\omega, 2_\omega\}| = 2, \\ 6, & \text{if } \tau \in \mathcal{D}^{(t)} \text{ and } \omega \in \mathcal{D}^{(t)} \text{ and } |\{1_\tau, 2_\tau\} \cap \{1_\omega, 2_\omega\}| = 2, \\ 7, & \text{if } \tau \in \mathcal{A}^{(t)} \text{ and } \omega \in \mathcal{D}^{(t)} \text{ and } |\{1_\tau, 2_\tau\} \cap \{1_\omega, 2_\omega\}| = 2, \\ 8, & \text{if } \tau \in \mathcal{D}^{(t)} \text{ and } \omega \in \mathcal{A}^{(t)} \text{ and } |\{1_\tau, 2_\tau\} \cap \{1_\omega, 2_\omega\}| = 2, \\ 9, & \text{otherwise (i.e., } |\{1_\tau, 2_\tau\} \cap \{1_\omega, 2_\omega\}| = 0). \end{cases} \quad (36)$$

THEOREM 5 (VARIANCE OF THINKD_{FAST}). Let $n_i^{(t)}$ be the number of Type- i triangle pairs in $\mathcal{A}^{(t)} \cup \mathcal{D}^{(t)}$. Likewise, Let $n_i^{(t)}[u]$ be the number of Type- i triangle pairs in $\mathcal{A}^{(t)}[u] \cup \mathcal{D}^{(t)}[u]$. Then,

$$\begin{aligned} Var[\bar{c}^{(t)}] &= (|\mathcal{A}^{(t)}| + |\mathcal{D}^{(t)}|) \cdot \frac{1 - r^2}{r^2} \\ &\quad + (n_1^{(t)} + n_2^{(t)} - n_3^{(t)} - n_4^{(t)}) \cdot \frac{1 - r}{r} \\ &\quad + (n_5^{(t)} + n_6^{(t)} - n_7^{(t)} - n_8^{(t)}) \cdot \frac{1 - r^2}{r^2}, \quad \forall t \geq 1. \end{aligned} \quad (37)$$

Likewise,

$$\begin{aligned} Var[c^{(t)}[u]] &= (|\mathcal{A}^{(t)}[u]| + |\mathcal{D}^{(t)}[u]|) \cdot \frac{1 - r^2}{r^2} \\ &\quad + (n_1^{(t)}[u] + n_2^{(t)}[u] - n_3^{(t)}[u] - n_4^{(t)}[u]) \cdot \frac{1 - r}{r} \\ &\quad + (n_5^{(t)}[u] + n_6^{(t)}[u] - n_7^{(t)}[u] - n_8^{(t)}[u]) \cdot \frac{1 - r^2}{r^2}, \quad \forall t \geq 1, \quad \forall u \in \mathcal{V}^{(t)}. \end{aligned} \quad (38)$$

PROOF. For each time $t \geq 1$, let $X^{(t)}$ be the random number in *Bernoulli*(r) drawn in line 12 of Algorithm 2 while the t -th element $e^{(t)}$ is processed. Then, from Lemma 4,

$$\{u, v\} \in \mathcal{S}^{(t)} \iff X^{(t)}_{uv} = 1. \quad (39)$$

Now, for each $\tau = (\{u, v, w\}, s) \in \mathcal{A}^{(t)} \cup \mathcal{D}^{(t)}$, let γ_τ be the amount of change in each of $\bar{c}$, $c[u]$, $c[v]$, and $c[w]$ due to the discovery of τ in line 9 or line 10 of Algorithm 2. Let $\delta_\tau = +1$ when $\tau \in \mathcal{A}^{(t)}$, i.e., when the last edge is added, and let $\delta_\tau = -1$ when $\tau \in \mathcal{D}^{(t)}$, i.e., when the last edge is deleted. Let $\{u, v\}$ be the edge added or deleted at time s without loss of generality. Then, $\gamma_\tau = \frac{\delta_\tau}{r^2}$ if both $\{v, w\}, \{w, u\} \in \mathcal{S}^{(s)}$ and 0 otherwise. Hence, combined with Equation (39),

$$\gamma_\tau = \frac{\delta_\tau}{r^2} X^{(1_\tau)} X^{(2_\tau)}. \quad (40)$$

Then from the definitions of γ_τ , $\bar{c}^{(t)} = \sum_{\tau \in \mathcal{A}^{(t)} \cup \mathcal{D}^{(t)}} \gamma_\tau$, and its variance is

$$Var[\bar{c}^{(t)}] = \sum_{\tau \in \mathcal{A}^{(t)} \cup \mathcal{D}^{(t)}} Var[\gamma_\tau] + \sum_{\tau \neq \omega \in \mathcal{A}^{(t)} \cup \mathcal{D}^{(t)}} Cov[\gamma_\tau, \gamma_\omega]. \quad (41)$$

For the variance term in Equation (41), note first that applying that $X^{(1_\tau)}$ and $X^{(2_\tau)}$ are independent *Bernoulli*(r) to Equation (40) gives $\mathbb{E}[\gamma_\tau]$ as

$$\mathbb{E}[\gamma_\tau] = \mathbb{E}\left[\frac{\delta_\tau}{r^2} X^{(1_\tau)} X^{(2_\tau)}\right] = \frac{\delta_\tau}{r^2} \mathbb{E}[X^{(1_\tau)}] \mathbb{E}[X^{(2_\tau)}] = \delta_\tau. \quad (42)$$

Then, further applying $\delta_\tau^2 = 1$ and $(X^{(s)})^2 = X^{(s)}$ to Equation (40) and again applying that $X^{(1_\tau)}$ and $X^{(2_\tau)}$ are independent *Bernoulli*(r) give $\text{Var}[\gamma_\tau]$ as

$$\begin{aligned} \text{Var}[\gamma_\tau] &= \mathbb{E}[\gamma_\tau^2] - (\mathbb{E}[\gamma_\tau])^2 = \mathbb{E}\left[\frac{\delta_\tau^2}{r^4} X^{(1_\tau)} X^{(2_\tau)}\right] - \delta_\tau^2 = \frac{1}{r^4} \mathbb{E}[X^{(1_\tau)}] \mathbb{E}[X^{(2_\tau)}] - 1 \\ &= \frac{1 - r^2}{r^2}. \end{aligned}$$

Hence the variance term in Equation (41) is computed as

$$\sum_{\tau \in \mathcal{A}^{(t)} \cup \mathcal{D}^{(t)}} \text{Var}[\gamma_\tau] = \sum_{\tau \in \mathcal{A}^{(t)} \cup \mathcal{D}^{(t)}} \frac{1 - r^2}{r^2} = (|\mathcal{A}^{(t)}| + |\mathcal{D}^{(t)}|) \cdot \frac{1 - r^2}{r^2}. \quad (43)$$

For the covariance term in Equation (41), applying Equation (40) and Equation (42) and using the fact that all the $X^{(s)}$'s are independent and identically distributed as *Bernoulli*(r) and $(X^{(s)})^2 = X^{(s)}$ yield the $\text{Cov}[\gamma_\tau, \gamma_\omega]$ as

$$\begin{aligned} \text{Cov}[\gamma_\tau, \gamma_\omega] &= \mathbb{E}[\gamma_\tau \gamma_\omega] - \mathbb{E}[\gamma_\tau] \mathbb{E}[\gamma_\omega] = \mathbb{E}\left[\frac{\delta_\tau \delta_\omega}{r^4} X^{(1_\tau)} X^{(2_\tau)} X^{(1_\omega)} X^{(2_\omega)}\right] - \delta_\tau \delta_\omega \\ &= \delta_\tau \delta_\omega \left(\frac{1}{r^4} \mathbb{E}\left[\prod_{i \in \{1_\tau, 2_\tau\} \cup \{1_\omega, 2_\omega\}} X^{(i)}\right] - 1 \right) \\ &= \delta_\tau \delta_\omega \left(\frac{1}{r^4} \prod_{i \in \{1_\tau, 2_\tau\} \cup \{1_\omega, 2_\omega\}} \mathbb{E}[X^{(i)}] - 1 \right) = \delta_\tau \delta_\omega \left(\frac{r^{|\{1_\tau, 2_\tau\} \cup \{1_\omega, 2_\omega\}|}}{r^4} - 1 \right). \end{aligned}$$

Then $\delta_\tau \delta_\omega = 1$ if $\tau, \omega \in \mathcal{A}^{(t)}$ or $\tau, \omega \in \mathcal{D}^{(t)}$, and $\delta_\tau \delta_\omega = -1$ if $\tau \in \mathcal{A}^{(t)}, \omega \in \mathcal{D}^{(t)}$ or $\tau \in \mathcal{D}^{(t)}, \omega \in \mathcal{A}^{(t)}$. Hence $\text{Cov}[\gamma_\tau, \gamma_\omega]$ can be calculated as

$$\text{Cov}[\gamma_\tau, \gamma_\omega] = \begin{cases} \frac{1-r}{r}, & \text{if } \text{Type}_{(\tau, \omega)} = 1 \text{ or } 2, \\ -\frac{1-r}{r}, & \text{if } \text{Type}_{(\tau, \omega)} = 3 \text{ or } 4, \\ \frac{1-r^2}{r^2}, & \text{if } \text{Type}_{(\tau, \omega)} = 5 \text{ or } 6, \\ -\frac{1-r^2}{r^2}, & \text{if } \text{Type}_{(\tau, \omega)} = 7 \text{ or } 8, \\ 0, & \text{if } \text{Type}_{(\tau, \omega)} = 9. \end{cases}$$

Hence the covariance term in Equation (41) is computed as

$$\begin{aligned} \sum_{\tau \neq \omega \in \mathcal{A}^{(t)} \cup \mathcal{D}^{(t)}} \text{Cov}[\gamma_\tau, \gamma_\omega] &= (n_1^{(t)} + n_2^{(t)} - n_3^{(t)} - n_4^{(t)}) \cdot \frac{1-r}{r} \\ &\quad + (n_5^{(t)} + n_6^{(t)} - n_7^{(t)} - n_8^{(t)}) \cdot \frac{1-r^2}{r^2}. \end{aligned} \quad (44)$$

Hence applying Equation (43) and Equation (44) to Equation (41) gives

$$\begin{aligned} \text{Var}[\hat{c}^{(t)}] &= \sum_{\tau \in \mathcal{A}^{(t)} \cup \mathcal{D}^{(t)}} \text{Var}[\gamma_\tau] + \sum_{\tau \neq \omega \in \mathcal{A}^{(t)} \cup \mathcal{D}^{(t)}} \text{Cov}[\gamma_\tau, \gamma_\omega] \\ &= (|\mathcal{A}^{(t)}| + |\mathcal{D}^{(t)}|) \cdot \frac{1-r^2}{r^2} + (n_1^{(t)} + n_2^{(t)} - n_3^{(t)} - n_4^{(t)}) \cdot \frac{1-r}{r} \\ &\quad + (n_5^{(t)} + n_6^{(t)} - n_7^{(t)} - n_8^{(t)}) \cdot \frac{1-r^2}{r^2}, \end{aligned}$$

which completes the proof of Equation (37).

For Equation (38), replacing $\bar{c}^{(t)}$ by $c^{(t)}[u]$, $\mathcal{A}^{(t)}$ by $\mathcal{A}^{(t)}[u]$, and $\mathcal{D}^{(s)}$ by $\mathcal{D}^{(s)}[u]$ and repeating above give a proof. $\square$

C EXTENSIONS: TRIANGLE COUNTING IN FULLY DYNAMIC MULTIGRAPH STREAMS

C.1 Notations and Problem Definition

A *multigraph* is a graph with parallel edges; hence, two nodes can be connected by more than one edge. Specifically, we define the multiplicity of an edge $\{u, v\}$ by $M(u, v)$, which indicates a number of parallel edges between u and v ; then, a multigraph has at least one edge whose $M(u, v) \geq 2$. Meanwhile, we call a graph with no parallel edges (i.e., all $M(u, v) = 1$) a *simple graph*.

Similar to the simple graph case, a *fully dynamic multigraph stream* represents the sequence of changes in a multigraph $\mathcal{G}$, and we denote the stream by $(e^{(1)}, e^{(2)}, \dots)$. As in [De Stefani et al. 2017], we assume that a triple $e^{(t)} = (\{u, v\}, l, \delta)$ of an edge $\{u, v\}$, a label l , which differentiates parallel edges, and a sign $\delta \in \{+, -\}$ arrives at each time $t \in \{1, 2, \dots\}$. Moreover, we assume that only new edges not in $\mathcal{G}$ (which can be parallel edges with new labels) can be added, and only existing edges in $\mathcal{G}$ can be deleted.

There are two ways of counting triangles in a multigraph. One way is to add the weights of triangles. The *weight* of each triangle $\{u, v, w\} \subset V$ is defined as $M(u, v) \times M(v, w) \times M(w, u)$. The other way is to count triangles while ignoring edge multiplicity. In this article, we choose the former scheme, which fully reflects given information.

We address the problem of estimating the counts of global and local triangles in a fully dynamic multigraph stream, which is summarized by Problem 2. As in Section 3.2, 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 2 (Global and Local Triangle Counting in a Fully Dynamic Multigraph Stream).

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

C.2 Extension of THINKD_{FAST} and THINKD_{ACC} to a Fully Dynamic Multigraph Stream

We extend THINKD_{FAST} and THINKD_{ACC} to a fully dynamic multigraph stream by making two key changes while keeping the other parts the same. The overall process of THINKD_{FAST} and THINKD_{ACC} for triangle counting in a fully dynamic multigraph stream is provided in Algorithm 4.

The first key change is to store parallel edges with their labels. Given an edge deletion, if we delete a parallel edge without considering labels, edges can be “overdeleted” and thus the counts of triangles can be underestimated. Therefore, given an edge deletion, we delete the same edge with the same label from memory (if it is in memory) without deleting parallel edges with different labels.

The second key change is the amount of changes in our estimates per new or deleted triangle. For each new observed triangle $\{u, v, w\}$, we increase the corresponding estimates by $M(v, w) \times M(w, u)/r^2$ (in THINKD_{FAST}) or $M(v, w) \times M(w, u)/p(|\mathcal{E}|, n_b, n_g)$ (in THINKD_{ACC}). We decrease the corresponding estimates by the same amounts for each observed deleted triangle $\{u, v, w\}$.

ALGORITHM 4: Extensions of $\text{THINKD}_{\text{FAST}}$ and $\text{THINKD}_{\text{ACC}}$ to a Multigraph

Inputs: fully dynamic multigraph stream: $(e^{(1)}, e^{(2)}, \dots) = ((\{u, v\}, l, \delta), \dots)$,
 sampling probability: r (for $\text{THINKD}_{\text{FAST}}$), and
 memory budget: $k (\geq 2)$ (for $\text{THINKD}_{\text{ACC}}$)

Outputs: estimate of the global triangle count: $\bar{c}$
 estimates of the local triangle counts: $c[u]$ for each node u

```

1  $S \leftarrow \emptyset$  (for  $\text{THINKD}_{\text{FAST}}$ )
2  $|\mathcal{E}| \leftarrow 0, n_b \leftarrow 0, n_g \leftarrow 0$  (for  $\text{THINKD}_{\text{ACC}}$ )
3 for each element  $e^{(t)} = (\{u, v\}, l, \delta)$  in the input stream do
4    $\text{UPDATE}(\{u, v\}, l, \delta)$ 
5   if  $\delta = +$  then  $\text{INSERT}(\{u, v\}, l)$ 
6   else if  $\delta = -$  then  $\text{DELETE}(\{u, v\}, l)$ 
7 Procedure  $\text{UPDATE}(\{u, v\}, l, \delta)$ :  $\triangleright$  update global and local triangle counts
8   compute  $\hat{N}[u] \cap \hat{N}[v]$   $\triangleright \hat{N}[u]$  and  $\hat{N}[v]$  are obtained from current  $S$ 
9   for each common neighbor  $w \in \hat{N}[u] \cap \hat{N}[v]$  do
10    if  $\text{THINKD}_{\text{FAST}}$  then
11      if  $\delta = +$  then increase  $\bar{c}, c[u], c[v]$ , and  $c[w]$  by  $\frac{M(v, w)M(w, u)}{r^2}$ 
12      else if  $\delta = -$  then decrease  $\bar{c}, c[u], c[v]$ , and  $c[w]$  by  $\frac{M(v, w)M(w, u)}{r^2}$ 
13    else if  $\text{THINKD}_{\text{ACC}}$  then
14      if  $\delta = +$  then increase  $\bar{c}, c[u], c[v]$ , and  $c[w]$  by  $\frac{M(v, w)M(w, u)}{p(|\mathcal{E}|, n_b, n_g)}$ 
15      else if  $\delta = -$  then decrease  $\bar{c}, c[u], c[v]$ , and  $c[w]$  by  $\frac{M(v, w)M(w, u)}{p(|\mathcal{E}|, n_b, n_g)}$ 
16 Procedure  $\text{INSERT}(\{u, v\}, l)$ :
17   if  $\text{THINKD}_{\text{FAST}}$  then
18     if a random number in  $\text{Bernoulli}(r)$  is 1 then  $S \leftarrow S \cup \{(\{u, v\}, l)\}$ 
19   else if  $\text{THINKD}_{\text{ACC}}$  then
20      $|\mathcal{E}| \leftarrow |\mathcal{E}| + 1$ 
21     if  $n_b + n_g = 0$  then
22       if  $|S| < k$  then  $S \leftarrow S \cup \{(\{u, v\}, l)\}$ 
23       else if a random number in  $\text{Bernoulli}(k/|\mathcal{E}|)$  is 1 then
24         replace a random edge in  $S$  with  $(\{u, v\}, l)$ 
25     else if a random number in  $\text{Bernoulli}(n_b/(n_b + n_g))$  is 1 then
26        $S \leftarrow S \cup \{(\{u, v\}, l)\}, n_b \leftarrow n_b + 1$ 
27     else  $n_g \leftarrow n_g + 1$ 
28 Procedure  $\text{DELETE}(\{u, v\}, l)$ :
29   if  $\text{THINKD}_{\text{FAST}}$  then
30     if  $(\{u, v\}, l) \in S$  then  $S \leftarrow S \setminus \{(\{u, v\}, l)\}$ 
31   else if  $\text{THINKD}_{\text{ACC}}$  then
32      $|\mathcal{E}| \leftarrow |\mathcal{E}| - 1$ 
33     if  $(\{u, v\}, l) \in S$  then
34        $S \leftarrow S \setminus \{(\{u, v\}, l)\}, n_b \leftarrow n_b - 1$ 
35     else  $n_g \leftarrow n_g - 1$ 

```

Table 5. Summary of the Real-world Multigraph Streams Used in Our Experiments.
M: million, K: thousand

Name	#Nodes	#Edges	Type
Enron [Klimt and Yang 2004]	87.0K	1.13M	Email Network
DBLP [Ley 2002]	1.28M	18.2M	Author Collaboration
Wikipedia [Sun et al. 2016]	2.86M	19.3M	Communication
Movie [Barabási and Albert 1999]	382K	33.1M	Actor Collaboration

With the above proper changes, THINKD maintains unbiased estimates of the global and local triangle counts in a given fully dynamic multigraph stream. THINKD still requires sublinear memory, while the exact memory requirements depend on the sampling ratio r or the memory budget k . Note that we can apply the same techniques used in Section 4.4 and Section 4.5 to prove the accuracy and complexity of THINKD in a multigraph stream. Specifically, we can simply treat parallel edges with different labels as different edges and follow the same steps in the sections.

C.3 Experiments

We empirically validate the accuracy and speed of THINKD in order to show our extensions of THINKD_{FAST} and THINKD_{ACC} are successful. We describe our experimental settings in Section C.3.1. We explain the accuracy of THINKD and other methods in Section C.3.2 and the speed in Section C.3.3.

C.3.1 Experimental Settings. We ran all experiments on a machine with 2.67 GHz Intel Xeon E7-8837 CPUs and 1 TB memory (up to 32 GB was used). The real-world multigraphs used in our experiments are summarized in Table 5. From them, we created fully-dynamic multigraph streams as in Section 5.1, while fixing α (i.e., the ratio of deleted edges) to 20%. We used TRIEST_{FD}, which supports triangle counting in fully-dynamic multigraph streams, as the competitor of THINKD.

C.3.2 Accuracy (THINKD is More Accurate Than Its Best Competitor). We compared the accuracies of THINKD_{FAST}, THINKD_{ACC}, and TRIEST_{FD}, as we changed the ratio of stored edges at the end of each input stream from 5% to 40%. Each evaluation metric was averaged over 1,000 trials in all the datasets. As seen in Figure 8, THINKD_{ACC} consistently gave the best trade-off between space and accuracy, followed by THINKD_{FAST}. Specifically, within the same memory budget, THINKD_{ACC} was up to 2.7× and 2.5× **more accurate** than TRIEST_{FD} in terms of global error and RMSE, respectively.

C.3.3 Speed (THINKD is Faster Than Its Best Competitor). We compared the speeds and accuracies of THINKD_{FAST}, THINKD_{ACC}, and TRIEST_{FD}. The detailed settings were the same as those in Section C.3.2, and we ignored time taken to wait for the arrival of elements to measure the speeds of the algorithms independently of the speed of the input stream. As shown in Figure 9, THINKD_{FAST} consistently gave the best trade-off between speed and accuracy, followed by THINKD_{ACC}. Specifically, for the same global error and RMSE, THINKD_{FAST} was up to 2.2× **faster** than TRIEST_{FD}.

D APPLICATION: DETECTING SUDDENLY EMERGING DENSE SUBGRAPHS

In this section, we apply THINKD to the task of detecting dense subgraphs created within a short time. Specifically, we first provide a short survey on dense-subgraph detection and sliding-window models. Then, we propose an algorithm called THINKD-SPOT, which utilizes THINKD to detect suddenly emerging dense subgraphs. Lastly, we experimentally show the advantages of THINKD-SPOT over state-of-the-art algorithms.

Fig. 8. **THINKD is accurate.** THINKD_{FAST} and THINKD_{ACC} provide a better trade-off between space and accuracy than TRIEST_{FD}. In particular, THINKD_{ACC} is up to 2.7× more accurate than TRIEST_{FD} within the same memory budget.

D.1 Survey on Detecting Dense Subgraphs and Sliding-Window Models

Detecting Dense Subgraphs. Theoretical work on identifying densest subgraphs includes exact and approximate algorithms for static graphs [Andersen and Chellapilla 2009; Bahmani et al. 2012; Charikar 2000; Goldberg 1984; Khuller and Saha 2009] and approximate algorithms for dynamic graphs [Bhattacharya et al. 2015; Epasto et al. 2015a; McGregor et al. 2015; Nasir et al. 2017]. We refer interested readers to a survey [Lee et al. 2010]. Unusually dense subgraphs in real-world graphs tend to signal anomalous or fraudulent behaviors, including a “follower-boosting” service on social media [Hooi et al. 2017; Jiang et al. 2014; Shin et al. 2018a], “copy-and-paste” bibliographies in citation networks [Prakash et al. 2010; Shin et al. 2018a], and websites that attempt to manipulate search engine rankings [Gibson et al. 2005]. Several recent algorithms detect dense subgraphs or more generally dense subtensors that are created within a short time or in a temporally synchronized manner [Beutel et al. 2013; Jiang et al. 2015; Maruhashi et al. 2011; Rozenshtein et al. 2017; Shin et al. 2018c, 2017a, 2017b]. Considering temporal aspects enables these algorithms to accurately identify suspicious subgraphs and subtensors, which indicate fake “Likes” on Facebook [Beutel et al. 2013], “edit wars” and bot activities on Wikipedia [Shin et al. 2018a, 2017b], network intrusions [Maruhashi et al. 2011; Shin et al. 2017a], and so on.

Fig. 9. **THINKD is fast.** THINKD_{FAST} and THINKD_{ACC} provide a better trade-off between speed and accuracy than TRIEST_{FD}. In particular, THINKD_{FAST} is up to 2.2× faster than TRIEST_{FD} when they are similarly accurate.

Sliding-Window Models. Our dense-subgraph detection method, described in the following section, is in the *timestamp-based sliding-window model* [Babcock et al. 2002], where the elements (or edges) whose arrival timestamp is within a time interval ΔT of the current time are maintained, while the others are “expired” and removed. This model is different from but similar to the *sequence-based sliding window model* [Datar et al. 2002], where the most recent k elements (or edges) are maintained, regardless of their arrival timestamps, while the others are removed. In addition to sampling algorithms in both models [Babcock et al. 2002], several graph algorithms in the latter model have been developed for connectivity tests, graph sparsification, minimum spanning forest detection, and so on [Crouch et al. 2013; McGregor 2014]. Note that THINKD is in the fully-dynamic graph stream model, which is more general than both sliding-windows models. Designing triangle counting algorithm specific to the sliding-windows models remains to be investigated as future work.

D.2 Proposed Method: THINKD-SPOT

We present an algorithm called THINKD-SPOT, which utilizes THINKD to detect dense subgraphs that are created within a short time. THINKD-SPOT, which is described in Figure 10, tracks the

Fig. 10. Pictorial description of THINKD-SPOT, which spots suddenly emerging dense subgraphs using THINKD. THINKD-SPOT maintains the estimated counts of global and local triangles created within ΔT time units. Spikes in the estimated count of global triangles indicate the emergence of sudden dense subgraphs, and the estimated counts of local triangles indicate the amount of contribution of each node to the sudden dense subgraphs.

estimated counts of triangles using THINKD and uses the sudden increases in the counts as the signal of suddenly emerging dense subgraphs.

Consider a graph whose edges have actual timestamps (rather than logical timestamps), and for each edge e , let t_e be its timestamp. THINKD-SPOT first creates a fully dynamic graph stream with deletions as follows:

- (1) For each edge e , create an addition $(e, +)$ with timestamp t_e ,
- (2) For each edge e , create a deletion $(e, -)$ with timestamp $t_e + \Delta T$,
- (3) Sort the additions and deletions from the previous steps in the increasing order of their timestamps.

Then, THINKD-SPOT processes the created stream using THINKD (either THINKD_{FAST} or THINKD_{ACC}). This makes THINKD-SPOT maintain the estimated counts of global and local triangles created within ΔT time units. THINKD-SPOT tracks the changes in the maintained estimate of the global triangle count, and if a spike with the estimate greater than a given threshold θ is encountered, THINKD-SPOT reports the emergence of a sudden dense subgraph. At each spike, THINKD-SPOT returns the maintained estimates of local triangle counts as the amount of contribution of each node to the sudden dense subgraph.

Although we describe THINKD-SPOT in an offline setting, it can be easily extended to online settings where new edges are added to the input graph while it is being processed. In online settings,

Datasets	Facebook		Epinion	
Measures	Running Time (milliseconds)	Precision @ Top-10	Running Time (milliseconds)	Precision @ Top-10
CROSSSPOT	54,975	0.00	11,173	0.00
D-CUBE	1,443	0.33	674	0.00
M-BIZ	1,298	0.31	667	0.41
DENSEALERT	12,080	1.00	1,843	0.59
THINKD-SPOT	617	1.00	223	0.75

(a) Running Time and Accuracy Averaged over 10 Runs.

Fig. 11. (a) **THINKD-SPOT is fast and accurate.** (b-e) The spikes in the estimated count of global triangles, obtained by THINKD-SPOT, indicate the injected suddenly emerging cliques more accurately than the spikes obtained by DENSEALERT. The arrows indicate the timestamps where the cliques are injected, and the stars indicate the false alarms in top-10 spikes. The triangles indicate false dismissals. (f-g) The nodes forming the injected cliques are clearly distinguished by their estimated counts of local triangles, obtained by THINKD-SPOT. The plots show the distributions of the estimated counts of local triangles at the top-10 spikes in (b-e).

to maintain the estimated counts of triangles created within ΔT time units, THINKD-SPOT processes the insertion of each new edge as soon as the edge arrives, while it processes the deletion of the edge after ΔT time units. While new edges need to be stored until they are removed after ΔT units, since they are written and deleted sequentially, they can be stored on disks.

D.3 Experiments

For empirical evaluation of THINKD-SPOT, we measured how rapidly and accurately THINKD-SPOT detects suddenly emerging dense subgraphs compared to its best competitors: DENSEALERT [Shin et al. 2017b], M-Biz [Shin et al. 2018c], D-CUBE [Shin et al. 2017a], and CROSSSPOT [Jiang et al. 2015], all of which are designed for detecting dense subgraphs or more generally dense subtenors

created within a short time.¹² Specifically, we randomly injected 10 cliques created within a day into the Facebook and Epinion datasets.¹³ That is, the timestamps of all edges composing each clique were within a day. The injected cliques were of size 6 to 15. Then, we measured the running time of each method and the precision at the top-10 outputs obtained by each method (i.e., the ratio of outputs that correspond to the injected cliques).

As summarized in Figure 11(a), THINKD-SPOT was the fastest and the most accurate in both datasets. Especially, THINKD-SPOT was $8.3 - 19.6\times$ faster than DENSEALERT, which was the second most accurate method. Figures 11(b)-(e) show that the spikes in the estimated count of global triangles obtained by THINKD-SPOT indicated the injected cliques more accurately than the spikes obtained by DENSEALERT. Figures 11(e)-(f) show that the nodes forming the injected cliques are clearly distinguished by their estimated counts of local triangles, obtained by THINKD-SPOT.

REFERENCES

- Nesreen K. Ahmed, Nick Duffield, Jennifer Neville, and Ramana Kompella. 2014. Graph sample and hold: A framework for big-graph analytics. In *KDD'14*.
- 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.
- Reid Andersen and Kumar Chellapilla. 2009. Finding dense subgraphs with size bounds. In *WAW'09*.
- Brian Babcock, Mayur Datar, and Rajeev Motwani. 2002. Sampling from a moving window over streaming data. In *SODA'02*.
- Bahman Bahmani, Ravi Kumar, and Sergei Vassilvitskii. 2012. Densest subgraph in streaming and mapreduce. *PVLDB* 5, 5 (2012), 454–465.
- Ziv Bar-Yossef, Ravi Kumar, and D. Sivakumar. 2002. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *SODA'02*.
- Albert-László Barabási and Réka Albert. 1999. Emergence of scaling in random networks. *Science* 286, 5439 (1999), 509–512.
- Vladimir Batagelj and Matjaž Zveršnik. 2007. Short cycle connectivity. *Discrete Mathematics* 307, 3 (2007), 310–318.
- Luca Becchetti, Paolo Boldi, Carlos Castillo, and Aristides Gionis. 2010. Efficient algorithms for large-scale local triangle counting. *TKDD* 4, 3 (2010), 13.
- Jonathan W. Berry, Bruce Hendrickson, Randall A. LaViolette, and Cynthia A. Phillips. 2011. Tolerating the community detection resolution limit with edge weighting. *Physical Review E* 83, 5 (2011), 056119.
- Alex Beutel, Wanhong Xu, Venkatesan Guruswami, Christopher Palow, and Christos Faloutsos. 2013. Copycatch: Stopping group attacks by spotting lockstep behavior in social networks. In *WWW'13*.
- Sayan Bhattacharya, Monika Henzinger, Danupon Nanongkai, and Charalampos Tsourakakis. 2015. Space-and time-efficient algorithm for maintaining dense subgraphs on one-pass dynamic streams. In *STOC'15*.
- Paul G. Brown and Peter J. Haas. 2006. Techniques for warehousing of sample data. In *ICDE'06*.
- Moses Charikar. 2000. Greedy approximation algorithms for finding dense components in a graph. In *APPROX'00*.
- Shumo Chu and James Cheng. 2012. Triangle listing in massive networks. *TKDD* 6, 4 (2012), 17.
- Michael S. Crouch, Andrew McGregor, and Daniel Stubbs. 2013. Dynamic graphs in the sliding-window model. In *ESA'13*. Springer, 337–348.
- Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. 2002. Maintaining stream statistics over sliding windows. *SICOMP* 31, 6 (2002), 1794–1813.
- Lorenzo De Stefani, Alessandro Epasto, Matteo Riondato, and Eli Upfal. 2017. TRIEST: Counting local and global triangles in fully dynamic streams with fixed memory size. *TKDD* 11, 4 (2017), 43.
- Alessandro Epasto, Silvio Lattanzi, Vahab Mirrokni, Ismail Oner Sebe, Ahmed Taei, and Sunita Verma. 2015b. Ego-net community mining applied to friend suggestion. *PVLDB* 9, 4 (2015), 324–335.
- Alessandro Epasto, Silvio Lattanzi, and Mauro Sozio. 2015a. Efficient densest subgraph computation in evolving graphs. In *WWW'15*.

¹²We implemented all the algorithms commonly in Java. We used THINKD_{ACC} with $k = 2000$ for triangle counting in THINKD-SPOT, and we set ΔT to a day for both THINKD-SPOT and DENSEALERT. We used the geometric average degree as the density measure in M-Biz and D-CUBE, since this measure led to the highest accuracy. Using each of M-Biz, D-CUBE, and CrossSpot, we detected 10 dense subgraphs. The other experimental settings were the same as those described in Section 5.1.

¹³We used the Facebook and Epinion datasets because the edges in them have timestamps. We ignored the edges with missing timestamps.

- Rainer Gemulla, Wolfgang Lehner, and Peter J. Haas. 2008. Maintaining bounded-size sample synopses of evolving datasets. *The VLDB Journal* 17, 2 (2008), 173–201.
- David Gibson, Ravi Kumar, and Andrew Tomkins. 2005. Discovering large dense subgraphs in massive graphs. In *VLDB'05*.
- Andrew V. Goldberg. 1984. *Finding a Maximum Density Subgraph*. Technical Report.
- Bronwyn H. Hall, Adam B. Jaffe, and Manuel Trajtenberg. 2001. *The NBER Patent Citation Data File: Lessons, Insights and Methodological Tools*. Technical Report. National Bureau of Economic Research.
- Guyue Han and Harish Sethu. 2017. Edge sample and discard: A new algorithm for counting triangles in large dynamic graphs. In *ASONAM'17*.
- Paul W. Holland and Samuel Leinhardt. 1971. Transitivity in structural models of small groups. *Comp. Group Stud* 2, 2 (1971), 107–124.
- Bryan Hooi, Kijung Shin, Hyun Ah Song, Alex Beutel, Neil Shah, and Christos Faloutsos. 2017. Graph-based fraud detection in the face of camouflage. *TKDD* 11, 4 (2017), 44.
- Xiaocheng Hu, Yufei Tao, and Chin-Wan Chung. 2014. I/O-efficient algorithms on triangle listing and counting. *TODS* 39, 4 (2014), 27.
- Madhav Jha, Comandur Seshadhri, and Ali Pinar. 2013. A space efficient streaming algorithm for triangle counting using the birthday paradox. In *KDD'13*.
- Meng Jiang, Alex Beutel, Peng Cui, Bryan Hooi, Shiqiang Yang, and Christos Faloutsos. 2015. A general suspiciousness metric for dense blocks in multimodal data. In *ICDM'15*.
- Meng Jiang, Peng Cui, Alex Beutel, Christos Faloutsos, and Shiqiang Yang. 2014. Catchsync: Catching synchronized behavior in large directed graphs. In *KDD'14*.
- Minsoo Jung, Sunmin Lee, Yongsub Lim, and U. Kang. 2016. FURL: Fixed-memory and uncertainty reducing local triangle counting for graph streams. *Data Mining and Knowledge Discovery* 33, 5 (2019), 1225–1253.
- Andreas Kemper. 2009. *Valuation of Network Effects in Software Markets: A Complex Networks Approach*. Springer Science & Business Media.
- Samir Khuller and Barna Saha. 2009. On finding dense subgraphs. In *ICALP'09*. 597–608.
- Bryan Klimt and Yiming Yang. 2004. Introducing the enron corpus. In *CEAS'04*.
- Mihail N. Kolountzakis, Gary L. Miller, Richard Peng, and Charalampos E. Tsourakakis. 2010. Efficient triangle counting in large graphs via degree-based vertex partitioning. In *WAW'10*.
- Konstantin Kutzkov and Rasmus Pagh. 2014. Triangle counting in dynamic graph streams. In *SWAT'14*.
- Victor E. Lee, Ning Ruan, Ruoming Jin, and Charu Aggarwal. 2010. A survey of algorithms for dense subgraph discovery. In *Managing and Mining Graph Data*. 303–336.
- Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. 2009. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics* 6, 1 (2009), 29–123.
- Michael Ley. 2002. The DBLP computer science bibliography: Evolution, research issues, perspectives. In *SPIRE'02*. Springer, 1–10.
- Yongsub Lim, Minsoo Jung, and U. Kang. 2018. Memory-efficient and accurate sampling for counting local triangles in graph streams: from simple to multigraphs. *TKDD* 12, 1 (2018), 4.
- R. Duncan Luce and Albert D. Perry. 1949. A method of matrix analysis of group structure. *Psychometrika* 14, 2 (1949), 95–116.
- Koji Maruhashi, Fan Guo, and Christos Faloutsos. 2011. Multiaspectforensics: Pattern mining on large-scale heterogeneous networks with tensor analysis. In *ASONAM'11*.
- Paolo Massa and Paolo Avesani. 2005. Controversial users demand local trust metrics: An experimental study on epinions.com community. In *AAAI'05*.
- Andrew McGregor. 2014. Graph stream algorithms: A survey. *ACM SIGMOD Record* 43, 1 (2014), 9–20.
- Andrew McGregor, David Tench, Sofya Vorotnikova, and Hoa T Vu. 2015. Densest subgraph in dynamic graph streams. In *MFCs'14*.
- Alan Mislove, Massimiliano Marcon, Krishna P. Gummadi, Peter Druschel, and Bobby Bhattacharjee. 2007. Measurement and analysis of online social networks. In *IMC'07*.
- Muhammad Anis Uddin Nasir, Aristides Gionis, Gianmarco De Francisci Morales, and Sarunas Girdzijauskas. 2017. Fully dynamic algorithm for top-k densest subgraphs. In *CIKM'17*.
- Mark E. J. Newman. 2003. The structure and function of complex networks. *SIAM Review* 45, 2 (2003), 167–256.
- Rasmus Pagh and Charalampos E. Tsourakakis. 2012. Colorful triangle counting and a mapreduce implementation. *Inform. Process. Lett.* 112, 7 (2012), 277–281.
- 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.
- B. A. Prakash, M. Seshadri, A. Sridharan, S. Machiraju, and C. Faloutsos. 2010. Eigenspokes: Surprising patterns and community structure in large graphs. In *PAKDD'10*.

- Polina Rozenshtein, Nikolaj Tatti, and Aristides Gionis. 2017. Finding dynamic dense subgraphs. *TKDD* 11, 3 (2017), 27.
- Kijung Shin. 2017. WRS: Waiting room sampling for accurate triangle counting in real graph streams. In *ICDM'17*.
- Kijung Shin, Tina Eliassi-Rad, and Christos Faloutsos. 2018a. Patterns and anomalies in k-cores of real-world graphs with applications. *Knowl. Inf. Syst.* 54, 3 (2018), 677–710.
- Kijung Shin, Mohammad Hammoud, Euiwoong Lee, Jinoh Oh, and Christos Faloutsos. 2018b. Tri-Fly: Distributed estimation of global and local triangle counts in graph streams. In *PAKDD'18*.
- Kijung Shin, Bryan Hooi, and Christos Faloutsos. 2018c. Fast, accurate, and flexible algorithms for dense subtensor mining. *TKDD* 12, 3 (2018), 30.
- Kijung Shin, Bryan Hooi, Jisu Kim, and Christos Faloutsos. 2017a. D-Cube: Dense-block detection in terabyte-scale tensors. In *WSDM'17*.
- Kijung Shin, Bryan Hooi, Jisu Kim, and Christos Faloutsos. 2017b. DenseAlert: Incremental dense-subtensor detection in tensor streams. In *KDD'17*. ACM, 1057–1066.
- Kijung Shin, Jisu Kim, Bryan Hooi, and Christos Faloutsos. 2018d. Think before you discard: Accurate triangle counting in graph streams with deletions. In *ECML/PKDD'18*.
- Charles Spearman. 1904. The proof and measurement of association between two things. *AJP* 15, 1 (1904), 72–101.
- Jun Sun, Jérôme Kunegis, and Steffen Staab. 2016. Predicting user roles in social networks using transfer learning with feature transformation. In *ICDMW'16*. IEEE, 128–135.
- Siddharth Suri and Sergei Vassilvitskii. 2011. Counting triangles and the curse of the last reducer. In *WWW'11*.
- Kanat Tangwongsan, Aduri Pavan, and Srikanta Tirthapura. 2013. Parallel triangle counting in massive streaming graphs. In *CIKM'13*.
- Charalampos E Tsourakakis. 2008. Fast counting of triangles in large real networks without counting: Algorithms and laws. In *ICDM'08*.
- 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. *SNAM* 1, 2 (2011), 75–81.
- Charalampos E. Tsourakakis, U. Kang, Gary L. Miller, and Christos Faloutsos. 2009. Doulion: Counting triangles in massive graphs with a coin. In *KDD'09*.
- Bimal Viswanath, Alan Mislove, Meeyoung Cha, and Krishna P. Gummadi. 2009. On the evolution of user interaction in facebook. In *WOSN'09*.
- Jeffrey S. Vitter. 1985. Random sampling with a reservoir. *TOMS* 11, 1 (1985), 37–57.
- Pinghui Wang, Yiyan Qi, Yu Sun, Xiangliang Zhang, Jing Tao, and Xiaohong Guan. 2017. Approximately counting triangles in large graph streams including edge duplicates with a fixed memory usage. *PVLDB* 11, 2 (2017), 162–175.
- Stanley Wasserman and Katherine Faust. 1994. *Social Network Analysis: Methods and Applications*. Vol. 8. Cambridge University Press.
- Duncan J. Watts and Steven H. Strogatz. 1998. Collective dynamics of ‘small-world’ networks. *Nature* 393, 6684 (1998), 440–442.
- Jaewon Yang and Jure Leskovec. 2015. Defining and evaluating network communities based on ground-truth. *KAIS* 42, 1 (2015), 181–213.

Received July 2018; revised August 2019; accepted October 2019