Research Paper
Computational Methods for Differential Equations
http://cmde.tabrizu.ac.ir
Vol. *, No. *, *, pp. 1-13
DOI:10.22034/cmde.2025.65909.3060

Space-efficient algorithms for counting triangles in data streams using trained oracles

Hossein Jowhari* **and Arash Rahmati**
Faculty of Mathematics, Khajeh Nasir Toosi University of Technology, Tehran, Iran.

Abstract

In this paper we study data stream algorithms for approximating the number of triangles under the assumption that the algorithm has access to an oracle that answers certain queries about the input graph. Specifically we present algorithms that process the input graph given as a sequence of edges (or vertices) and output an estimate of the number of triangles in the given graph. We consider algorithms that, while processing the input stream, have access to a degree oracle (given a vertex, the oracle provides the degree of the queried vertex) or an edge-triangle oracle where the oracle answers whether an edge (u, v) participates in a triangle or not. We implement two single-pass algorithms and the associated oracles in both the edge-arrival and the vertex-arrival models, and evaluate their performance on real-world datasets. Despite the inaccuracies of the oracles used in our experiments, our study shows that they can improve the performance of state-of-the-art triangle counting algorithms on some real-world graphs.

Keywords. Counting triangles, Data stream model, Learning-Augmented algorithms.
2010 Mathematics Subject Classification. 68W27, 68T01, 05C30.

1. Introduction

Graphs are useful structures that are used to model real-world problems by representing relationships between entities. Computing the structural properties of a graph that models a real-world problem provides insight and facilitates the analysis of the problem. A key structural property of a graph, with many applications, is its number of triangles, i.e. the number of triplets of vertices in which each vertex is connected to the other two vertices. Algorithms for counting triangles in graphs have been studied extensively in the classical model of computations using combinatorial approaches [2] and linear algebraic methods [17, 27]. This problem has also been studied in alternative models of computation such as the data stream model [3, 10, 11, 16, 22] or sampling-based frameworks [8, 23]. In this paper, we focus on the data stream model of computation.

In the data stream model [18, 19], the input graph is given as a sequence of edges and the algorithm is allowed to have one pass (or few passes) over the input while having memory restrictions. Since the space usage of the algorithm is small in comparison with the input size (number of edges), the algorithms in this model often resort to randomness and approximation to cope with the strict memory limitations.

The problem of counting the number of triangles in a graph presented as a stream of edges was first introduced in 2002 [3], and has since been extensively studied in the streaming model due to its wide range of applications, including spam detection, community mining, and link prediction [1]. The problem has been studied under the assumption of both single-pass [10] and multi-pass algorithms [4]. Some researchers have focused on the insertion-only model where edges are not deleted later on once they appear in the stream [16, 22], whereas others have worked on the dynamic setting [25, 26], which handles edge deletions as well. The order of the edges is divided into three main categories: arbitrary order, random order, and vertex arrival. The most challenging case is the arbitrary order where edges may arrive in any order that is decided by an adversary. In the random order model, as the name suggests, the edges

Received: 16 February 2025 ; Accepted: 22 May 2025.
* Corresponding author. Email: jowhari@kntu.ac.ir.

randomly appear in the stream. Finally, in the vertex-arrival model, all edges incident to a certain vertex appear together and thus every edge is seen twice in the stream.

Recently, the problem of counting triangles has been explored within the framework of learning-augmented algorithms. In this context, it is assumed that a (noisy) predictor or oracle is available which is capable of answering specific queries about the input, while the data stream is processed. In practice, the predictor is constructed by training a machine learning model on previous instances of the problem or by utilizing other features of the input. In some cases, a previously stored instance of the problem can directly fulfill the role of the predictor. Although the predictor may give incorrect or approximate information about the input, assuming its error is bounded, empirical results demonstrate that trained oracles enhance algorithm efficiency.

In this direction, Chen et al. [6] initiated the study of learning-augmented algorithms for estimating the number of triangles in the data stream model. They proposed one-pass algorithms that leverage a heavy-edge oracle, which predicts whether a given edge participates in many triangles. Specifically, they demonstrated that access to such an oracle enables space bounds that are unattainable without this assumption. Additionally, they presented experimental results on real-world datasets, highlighting the practicality of their approach.

Inspired by Chen et al.'s work, in this paper we study a related oracle called an "edge-triangle oracle" which decides whether an edge participates in at least one triangle or not. We also consider the "degree oracle" which provides the degree of a vertex once asked (the degree oracle has been considered before in a work by McGregor et al. [16]). We revisit prior algorithms and observe that, in certain cases, the space complexity can be reduced—or the number of passes decreased—when either an edge-triangle oracle or a degree oracle is available (or both). Our theoretical results are detailed in section 1.3. On the experimental side, we demonstrate that the edge-triangle oracle can offer practical benefits. Specifically, our experiments show that, in both the edge-arrival and vertex-arrival models, augmenting the framework of [6] with an edge-triangle oracle yields more accurate triangle count estimates in low-memory settings on certain datasets.

1.1. **Preliminaries.** Given a graph $G(V, E)$ with m edges and n vertices, we focus on (ϵ, δ)-estimation of $T(G)$, the number of triangles in the graph using randomized algorithms with high probability. In other words, we need to guarantee $\mathbb{P}[|\hat{T} - T(G)| > \epsilon T(G)] < \delta$ where $\hat{T}$ is our estimate. We only work on the insertion-only model in this paper. Let $\Delta(G)$, Δ_E, and Δ_V denote the maximum degree of G, the maximum number of triangles on any given edge, and the maximum number of triangles on any given vertex respectively. Now we give formal definition of the oracles used in the previous works and the Edge-Triangle oracle introduced in this work.

Definition 1.1 (Degree Oracle)**.** Given a vertex u, a degree oracle, Deg-Oracle(.), returns the degree of the vertex, i.e. Deg-Oracle$(u) = d_u$.

Definition 1.2 (Heavy-Edge Oracle)**.** If t_e denotes the number of triangles incident to the edge $e = \{u, v\}$, a heavy-edge oracle, Heavy-Oracle(e), returns TRUE if $t_e > \theta$ and FALSE otherwise, where θ is the heaviness threshold.

Definition 1.3 (Edge-Triangle Oracle)**.** Given an edge e, Edge-Triangle-Oracle(e), returns TRUE if $t_e > 0$ and FALSE otherwise.

In our algorithmic analysis, we make use of Chebyshev's inequality and Chernoff bounds, which we assume the reader is familiar with. More specifically, we rely on the following standard application of these concentration inequalities, known as the Median-of-Means estimator. A detailed proof of correctness can be found in [5].

Lemma 1.4 (Median-of-Means Estimator)**.** *There is a universal positive constant c such that the following holds. Let the random variable X be an unbiased estimator for a real quantity Q. Let $\{X_{ij}\}_{i\in[t],j\in[k]}$ be a collection of independent random variables with each X_{ij} distributed identically to X, i.e. $\mathbb{E}[X_{ij}] = Q$, where $t = c \log \frac{1}{\delta}$ and $k = \frac{3\,\mathrm{Var}[X]}{\epsilon^2(\mathbb{E}[X])^2}$. Let $Z = \mathit{median}_{i\in[t]} \left(\frac{1}{k} \sum_{j=1}^{k} X_{ij} \right)$. Then, we know Z is an (ϵ, δ)-estimate for Q as stated below.*

$$\mathbb{P}\left(|Z - Q| \geq \epsilon Q\right) \leq \delta.$$

Thus, if an algorithm can produce X using s bits of space, then there is an (ϵ, δ)-estimation algorithm using the following bits of space.

$$O\left(s \cdot \frac{\mathrm{Var}[X]}{(\mathbb{E}[X])^2} \cdot \frac{1}{\epsilon^2} \log \frac{1}{\delta}\right).$$

1.2. **Previous Work.** We summarize the space bounds of existing algorithms under arbitrary order streams for both one-pass and multi-pass settings in Table 1 and Table 2, respectively. We use $\tilde{O}$ notation to suppress logarithmic factors, which commonly arise in randomized algorithms due to independent repetitions. For simplicity, we denote the number of triangles as T rather than $T(G)$, as the reference to the input graph is clear from context.

TABLE 1. Previous *one-pass, arbitrary order* triangle counting algorithms.

Ref.	Year	Space	Oracle
[11]	2005	$\tilde{O}(\epsilon^{-2}m\Delta^2/T)$	-
[20]	2012	$\tilde{O}(m(\epsilon^{-2}\Delta_E/T + \varepsilon^{-1}/\sqrt{T}))$	-
[22]	2013	$\tilde{O}(\epsilon^{-2}m\Delta/T)$	-
[12]	2018	$\tilde{O}(\epsilon^{-1}m\sqrt{\Delta_E/T})$	-
		$\tilde{O}(\epsilon^{-1}m\sqrt{\Delta_V/T})$	-
[10]	2021	$\tilde{O}(\epsilon^{-2}(m/T)(\Delta_E + \sqrt{\Delta_V}))$	-
[6]	2022	$\tilde{O}(\epsilon^{-1}(m/\sqrt{T} + \sqrt{m}))$	heavy-edge

Apart from the dependencies on T and m, we observe that all prior algorithms—except for [6]—also depend on Δ_E (note that $\Delta_E \leq \Delta$ and $\Delta_E \leq \Delta_V$). It is noteworthy that, the most recent work [6] eliminates this dependency by leveraging a heavy-edge oracle. In their setting, a heavy edge is defined as one that participates in at least θ triangles.

TABLE 2. Space bound of the previous *multi-pass, arbitrary order* triangle counting algorithms.

Ref.	Year	Space	Pass	Oracle
[7]	2017	$\tilde{O}(\epsilon^{-2.5}m/\sqrt{T})$	2	-
[16]	2016	$\tilde{O}(\epsilon^{-2}m/\sqrt{T})$	2	-
		$\tilde{O}(\epsilon^{-2}m^{3/2}/T)$	3	degree
[4]	2017	$\tilde{O}(\epsilon^{-2}m^{3/2}/T)$	4	-
[9]	2022	$\tilde{O}(\epsilon^{-2}m^{3/2}/T)$	3	-

We point out that the algorithm given by Bera and Chakrabarti [4] runs in 3 passes if given access to a degree oracle. Note that depending on whether $T > m$ or not, an algorithm with space complexity $m^{3/2}/T$ may outperform an algorithm with complexity $m/\sqrt{T}$, or vice versa. Interestingly, Fichtenberger et al. [9] showed an algorithm that achieves the space complexity of McGregor et al. [16] and runs in the same number of passes but without relying on a degree oracle. We also note that the algorithms presented in [4, 9] are general frameworks that can be applied to counting arbitrary subgraphs, particularly odd cycles. However, for the purpose of comparison with our triangle counting algorithms, we report only the space complexities of their methods when specialized to the triangle counting problem, which is the primary focus of this paper.

1.3. **Our Results.** Our results are of both theoretical and practical interest. Theoretical bounds for our algorithms are summarized in Table 3, where we highlight the space complexity of each algorithm along with its corresponding oracle usage. On the experimental side, we evaluate a modified version of the single-pass algorithms from [6], in which we incorporate an oracle that identifies and discards *unimportant* edges—those not involved in any triangle.

This oracle operates either by directly referencing earlier instances of the graph or by using machine learning models trained to predict such edges. Our experimental results demonstrate that edge-triangle oracles, which directly utilize previous instances of the graph to identify unimportant edges, are sufficiently accurate to achieve both lower relative error and reduced variance compared to the one-pass streaming algorithm of Chen et al. However, when we use machine learning models to train oracles on the specific datasets such as Reddit Hyperlinks, or when we use the first half of the edges as our oracle in the large Wiki-talk dataset, the resulting oracles do not provide enough accuracy to improve our estimates of T.

TABLE 3. Triangle counting algorithms in this work.

Space	Pass	Order	Oracle
$\tilde{O}(\epsilon^{-2}\Delta)$	1	arbitrary	edge-triangle
$\tilde{O}(\epsilon^{-2}m^{3/2}/T)$	2	arbitrary	degree
$\tilde{O}(\epsilon^{-2}\sqrt{m})$	2	arbitrary	edge-triangle + degree

2. ALGORITHMS

In this section we present our algorithms for the triangle counting problem that assume the existence of an oracle.

2.1. An algorithm with a degree oracle.

Assuming a degree oracle, we build up on the algorithm of [4] using the concepts in [22]. Vertices are compared based on their degrees; if two vertices have equal degrees, their names are compared lexicographically. Let $\prec$ denote this ordering relation. Let T_e be the number of triangles on edge e where the maximum degree of the triangle is not on this edge. More precisely,

$$T_{e=\{u,v\}} = \{s \in V(G) \mid (\{s,u\} \in E(G)) \ \wedge \ (\{s,v\} \in E(G)) \ \wedge \ \max\{u,v\} \prec s\}.$$

Under this ordering, it holds that $\sum_{e\in E(G)} T_e = T$, and it has been shown in [4] that for all edges e, we have $T_e \leq \sqrt{2m}$, reliant on the fact that the number of neighbors w of any fixed vertex v with higher degrees, i.e. $v \prec w$, is bounded by $\sqrt{2m}$. Based on this, we now present Algorithm 1.

Algorithm 1 $\tilde{O}(\varepsilon^{-2}m^{3/2}/T)$-Space, 2-Pass Algorithm.

Pass 1:
Select one edge $e_1 = \{u, v\}$ using reservoir sampling.
Pass 2:
WLOG, we assume $v \prec u$ using the degree oracle
$(Y, d', flag, e_2) \leftarrow (0, 0, 0, \emptyset)$
for edges in the form of $e_i = \{u, w_i\}$ or $e_i = \{v, w_i\}$ **do**
  if $u \prec w_i$ **then**
    $d' \leftarrow d' + 1$
    if coin$(1/d')$= *"head"* **then**
      $(e_2, flag) \leftarrow (e_i, 0)$
    else
      if e_i completes the wedge e_1e_2 **then**
        $flag \leftarrow 1$
if $flag = 1$ **then** $Y \leftarrow d'$
$X \leftarrow mY$
return X

In Algorithm 1, the outcome of coin(p) is "*head*" with probability p. In the first pass, the algorithm uniformly samples one edge at random using reservoir sampling [28]. In the second pass, the algorithm samples one neighbor

of the edge (u, v). However, the degree oracle helps us sample what we call a "good" neighbor: a neighbor with a higher degree than those of u and v. Note that in the algorithm in [4], one has to sample multiple neighbors to detect a "good" neighbor and thus the space complexity of an independent repetition in [4] becomes superconstant. Finally, the algorithm waits for the arrival of the completing edge of the triangle after w_i (the "good" neighbor) has been sampled. We will now calculate the expectation and the variance of the output random variable of algorithm 1 and prove theorem 2.1.

Theorem 2.1. *Given graph G as a stream of edges, there is a 2-pass algorithm (i.e. 1) that benefits from a degree oracle and approximates $T(G)$ within $1+\epsilon$ factor using $\tilde{O}(\frac{m^{3/2}}{\epsilon^2 T})$ space.*

Proof. If we assume that the algorithm has sampled $e_i = \{u, v\}$ in the first pass, we can calculate $\mathbb{E}[Y \mid e_i]$. Let $c'(e)$ be the number of *good* neighbors of $e = \{u, v\}$ that are higher than u and v in the ordering relation $\prec$. The algorithm, in the second pass, picks a fixed good neighbor of $e_i = \{u, v\}$ with probability $\frac{1}{c'(e_i)}$. Also $d' = c'(e_i)$. Thus expectation is calculated as follows.

$$\mathbb{E}[Y \mid e_i] = \frac{T_{e_i}}{c'(e_i)} c'(e_i) = T_{e_i}.$$

So,

$$\mathbb{E}[X] = m\mathbb{E}[Y] = m(\frac{1}{m}\mathbb{E}[Y \mid e_1] + \cdots + \frac{1}{m}\mathbb{E}[Y \mid e_m]) = m\frac{1}{m}\sum_{e \in E} T_e = T.$$

We now proceed to calculate an upper bound on the variance.

$$\mathrm{Var}[X] = \mathrm{Var}[mY] = m^2\mathrm{Var}[Y] \leq m^2\mathbb{E}[Y^2] = m^2\frac{1}{m}\sum_{e \in E}\mathbb{E}[Y^2|e] = m\sum_{e_i \in E}\frac{T_{e_i}}{c'(e_i)}c'(e_i)c'(e_i)$$

$$\leq m\max_{e_i \in E}\{c'(e_i)\}\sum_{e_i \in E} T_{e_i} = 2m\sqrt{2m}T = O(m^{3/2}T).$$

Now that we have the upper bound on the variance of our unbiased estimator and that we know each execution requires $O(1)$ space, we use lemma 1.4 to calculate the number of repetitions required as follows.

$$\text{Repetitions} = \frac{3\mathrm{Var}[X]}{\epsilon^2(\mathbb{E}[X])^2} c\log(\frac{1}{\delta}) = O(\frac{m^{3/2}T}{\epsilon^2 T^2} c\log(\frac{1}{\delta})) = \tilde{O}(\frac{m^{3/2}}{\epsilon^2 T}).$$

□

2.2. **An algorithm with an edge-triangle oracle.** Pavan et al. [22] proposed a single-pass streaming algorithm in the arbitrary-order model that approximates T using $\tilde{O}(\epsilon^{-2}m\Delta/T)$ space, achieving a multiplicative error guarantee. With access to an edge-triangle oracle, as defined in Definition 1.3, it is straightforward to extend their approach to prove the following theorem. Specifically, the only requirement is that the first edge r_1 in their algorithm be sampled from the set of "good" edges—those that participate in at least one triangle. In this case, the parameter m can be replaced by m', the number of triangle-participating edges. Since it holds that $m' \leq 3T$, this substitution leads directly to the following result.

Theorem 2.2. *Given graph G as a stream of edges, there is a 1-pass algorithm that benefits from an edge-triangle oracle and approximates $T(G)$ within $1+\epsilon$ factor using $\tilde{O}(\frac{\Delta(G)}{\epsilon^2})$ space.*

Proof. Follows from the algorithm in Pavan et. al. [22]. □

2.3. **A combination of two oracles.** In this section, we explore the power of having access to both oracles. Similar to what we proposed in section 2.1, we use the degree oracle to sample "good" neighbors. Furthermore, we sample a "good" edge in the first pass similar to section 2.2. The pseudocode of this algorithm is illustrated in Algorithm 2 where we assume $v \prec u$.

Algorithm 2 $\tilde{O}(\varepsilon^{-2}\sqrt{m})$-Space, 2-Pass Algorithm.

Pass 1:
$(m', e_1) \leftarrow (0, \emptyset)$
for edges e_i in the stream **do**
  if Edge-Triangle-Oracle$(e_i) =$ *"TRUE"* **then** ▷ e_i is in a triangle.
    $m' \leftarrow m' + 1$
    $e_1 = \{u, v\} \leftarrow e_i$ w.p. $(1/m')$
Pass 2:
WLOG, we assume $v \prec u$ using the degree oracle
$(Y, d', flag, e_2) \leftarrow (0, 0, 0, \emptyset)$
for edges in the form of $e_i = \{u, w_i\}$ or $e_i = \{v, w_i\}$ **do**
  if $u \prec w_i$ **then**
    $d' \leftarrow d' + 1$
    if coin$(1/d')$=*"head"* **then**
      $(e_2, flag) \leftarrow (e_i, 0)$
    else
      if e_i completes the wedge e_1e_2 **then**
        $flag \leftarrow 1$
if $flag = 1$ **then** $Y \leftarrow d'$
return $X = m'Y$

Theorem 2.3. *Given graph G as a stream of edges, there is a 2-pass algorithm (i.e. 2) that benefits from a degree oracle and an edge-triangle oracle, and approximates $T(G)$ within $1+\epsilon$ factor using $\tilde{O}(\epsilon^{-2}\sqrt{m})$ space.*

Proof. Note that, in the end of the second pass, m' equals the number of edges in the input graph that participate in at least one triangle. Using a similar discussion presented in Theorem 2.1, we see that:

$$\mathbb{E}[X] = T, \qquad \mathrm{Var}[X] = O(m'\sqrt{m}T) = O(\sqrt{m}T^2).$$

Here we have used the fact that $m' \leq 3T$. Since the space complexity of one execution of the algorithm is $O(1)$, the space complexity required to achieve $(1+\epsilon)$ approximation is

$$\mathsf{Repetitions} = \frac{3\mathrm{Var}[X]}{\epsilon^2(\mathbb{E}[X])^2} c\log(\frac{1}{\delta}) = O(\frac{\sqrt{m}T^2}{\epsilon^2 T^2} c\log(\frac{1}{\delta})) = \tilde{O}(\frac{\sqrt{m}}{\epsilon^2}).$$

□

3. Experimental Results

3.1. **Datasets.** We use four datasets described as follows.

- Oregon[1]: This dataset consists of 9 graphs $\{\#1, ..., \#9\}$ of Autonomous Systems (AS) peering information inferred from Oregon route-views between March 31 2001 and May 26 2001 on the internet [15].
- As-Caida[2]: The dataset contains 122 CAIDA Autonomous System (AS) graphs, from January 2004 to November 2007. Each file contains a full AS graph derived from a set of RouteViews BGP table snapshots. We will use the data from 2006 and 2007 in our experiments. There are 52 instances of the graph of 2006 $\{\#1, ..., \#52\}$, and there are 46 instances of the graph of 2007 $\{\#1, ..., \#46\}$.
- Reddit Hyperlinks[3]: The hyperlink network [13] represents the directed connections between two subreddits (a subreddit is a community on Reddit, a social network). Note that we ignore edge directions in our experiments. 300-dimensional embeddings are also available for most of the nodes. The network is extracted from publicly

[1] https://snap.stanford.edu/data/Oregon-1.html
[2] https://snap.stanford.edu/data/as-Caida.html
[3] https://snap.stanford.edu/data/soc-RedditHyperlinks.html

TABLE 4. Statistics of the graph datasets.

Dataset	n	m	T	Δ	Δ_E	Δ_V
OREGON#1	10670	22002	17144	2312	526	3431
CAIDA-2006#1	21202	42925	30433	2381	578	3530
CAIDA-2007#1	24013	49332	40475	2377	602	4590
Reddit-Hyperlink	35775	124330	406391	2336	725	31967
Wiki-talk#1	528230	1346354	3603493	71211	1833	117709

available Reddit data of 2.5 years from Jan 2014 to April 2017. The subreddit-to-subreddit hyperlink network is extracted from the posts that create hyperlinks from one subreddit to another. We say a hyperlink originates from a post in the source community and links to a post in the target community.

- Wiki-talk[1]: The Wiki-talk dataset [14, 21] is the largest dataset used in our experiments, containing over one million edges. It is a temporal graph representing interactions between Wikipedia users, where a directed edge (u, v, t) indicates that user u edited user v's talk page at time t. For our experiments, we treat the graph as undirected and remove duplicate edges.

The statistics of the first instances of Oregon and As-Caida along with the statistics of Reddit Hyperlinks and Wiki-talk are summarized in Table 4.

3.2. **How to make oracles.** We build oracles in three ways similar to [6].

- In Oregon and As-Caida, we directly use previous data. In fact, we look at the first instance of each graph and memorize those important (heavy) edges. We also memorize unimportant edges (edges not involved in any triangle).
- In Reddit Hyperlinks, since we have node representations, $f(u)$, we train machine learning models. More specifically, we train a linear regression model as our heavy-edge oracle, and a logistic regression model as our edge-triangle oracle that outputs yes/no. Similar to [6], the labels are calculated using the exact count, and the features of edges are made using node embeddings as follows.

$$f(e = \{u, v\}) = \underbrace{(f(u), f(v), \|f(u) - f(v)\|_1, \|f(u) - f(v)\|_2)}_{\text{602-dimensional vector}}.$$

- In the case of Wiki-talk, we divide the graph into two segments, referred to as Part #1 and Part #2. Since Wiki-talk is a temporal graph in which each edge is associated with a timestamp, we first sort the edges chronologically. The first half of the edges is used to construct an edge-triangle oracle using an exact triangle counting approach. The remaining half serves as the input stream over which we approximate the number of triangles.

3.3. **One-Pass Algorithms of Interest.** For practical considerations, the algorithms we implement and evaluate on our datasets differ from the theoretical algorithms analyzed in Section 2. While inspired by the same principles, the implemented versions are adapted to better suit real-world constraints such as space limits and oracle availability. We primarily compare three algorithms: the oracle-based algorithm of Chen et al., our proposed modification of their method, and the non-oracle ThinkD algorithm. The one-pass variant of ThinkD is widely regarded as state-of-the-art among non-oracle algorithms and is available in two versions: *Fast* and *Accurate*. In our experiments, we have implemented the *Accurate* version and report its performance across the datasets under the edge-arrival model.

We primarily compare our algorithms with Algorithms 1 and 4 from the work of Chen et al. [6]. This choice is motivated by their empirical results, which demonstrate that these oracle-based algorithms outperform prior state-of-the-art one-pass streaming algorithms without oracle access—such as ThinkD [25] and WRS [24]—on most datasets, including Oregon, Caida-2006, and Caida-2007, under both edge-arrival and vertex-arrival models. Thus, at least on these datasets, the algorithms of Chen et al. can be considered state-of-the-art. We emphasize that our proposed

[1] https://snap.stanford.edu/data/wiki-talk-temporal.html

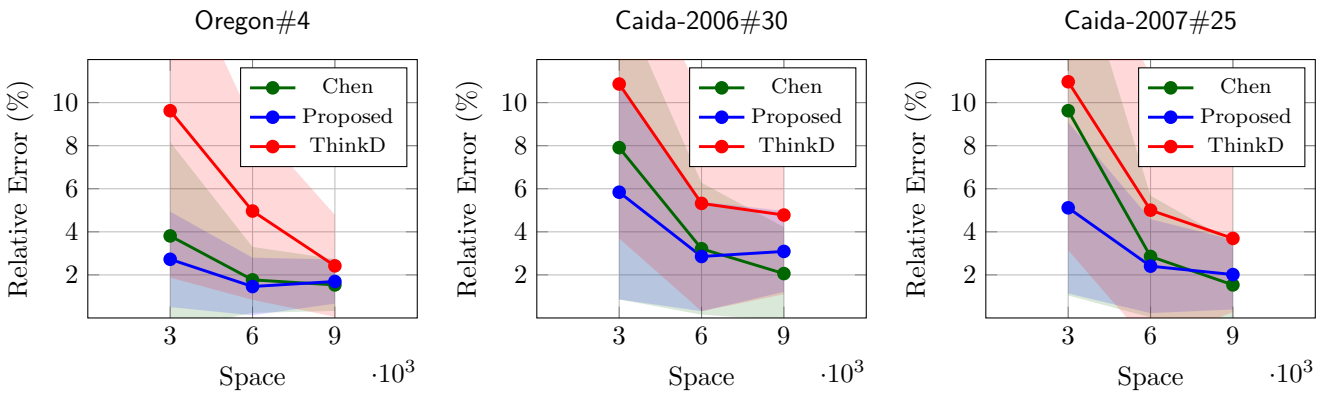

FIGURE 1. Comparison of algorithms (edge-arrival) on Oregon and CAIDA datasets.

algorithms build upon the framework introduced by Chen et al., particularly the use of heavy-edge oracles. In addition, we incorporate edge-triangle prediction oracles, which extend their approach. This enhancement—combined with the original strategy of retaining important (i.e., heavy) edges—not only improves the accuracy of our estimates but, more importantly, significantly reduces variance across multiple executions, especially in settings where only a limited number of edges can be sampled.

We refer the reader to [6] for a detailed description of Chen et al.'s algorithm. Briefly, their core idea is to leverage an oracle to retain heavy edges separately while uniformly sampling the remaining light edges. Triangles are then categorized based on the combination of heavy and light edges they contain, with each category assigned a distinct counter. These counters are appropriately scaled by the inverse of the sampling probability for each triangle type. Finally, the adjusted counts are summed to produce an estimate of the total number of triangles in the graph.

3.4. **Comparison of Algorithms.** We use Oregon#1, Caida-2006#1, and Caida-2007#1 as oracles to detect heavy and unimportant edges. The input of the algorithms are Oregon#4, Caida-2006#30, and Caida-2007#25. The error is $\left|(1-\hat{T}/T)\times 100\right|$. We run each algorithm 50 times and calculate the median relative error. The colors around each line is ± 1 standard deviation of the relative errors in 50 independent executions.

We observe from Figure 1 that the proposed algorithm in all cases has lower variance especially when the space is very limited. The median relative errors of the proposed algorithm, except for the space of 9000 edges, have been lower compared to the work of Chen et al. [6]. It is reasonable that the proposed algorithm cannot outperform the work of Chen et al. when we are allowed to sample many edges since the oracles are not perfect, and we lose some triangles by mistakenly removing some edges involved in triangles (see discussion 3.5 for more details). Despite a higher error at 9000 sampled edges, the variance of the proposed algorithm is still lower. We also note that the non-oracle baseline of ThinkD is noticeably outperformed by the oracle-based methods on these datasets.

For the Oregon dataset, we further evaluate the performance of the proposed algorithm across multiple instances, as shown in Figure 2. In this experiment, both algorithms are allocated the same memory budget, allowing up to 3000 edges. The vertical lines in Figure 2 indicate ± 1 standard deviation of the relative error, computed over 50 independent runs.

We have also compared our algorithm to that of Chen et al. (see Figure 3) on the Wiki-talk dataset. On this relatively bigger dataset, we have allowed the algorithm to keep as many as 30K edges. Similarly, we see our proposed algorithm outperforms the work of Chen et al. in the low memory setting. However, the non-oracle algorithm of ThinkD outperforms both Chen et al.'s and our proposed methods. In subsection 3.5, we will explain that this happens due to the lower accuracy of the oracles that are based on the first half of the edges in Wiki-talk graph.

To further evaluate our idea of removing unimportant edges, we implemented the first algorithm of [6] which is in the vertex arrival model. Vertex-arrival model is a simpler setting in which vertices arrive with their neighbors one at

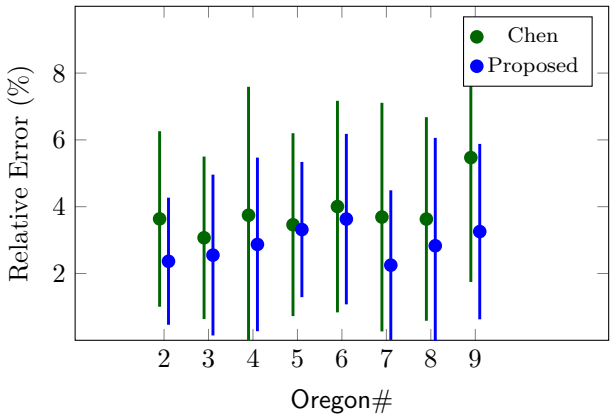

FIGURE 2. Comparison of algorithms on other instances of Oregon (edge-arrival).

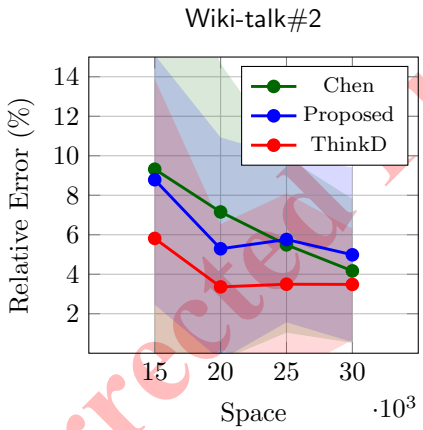

FIGURE 3. Comparison (edge-arrival) on Wiki-talk dataset.

a time. The errors become smaller in this setting and using our idea of removing unnecessary edges makes the errors even smaller as seen in Figure 4 for datasets Oregon#4, CAIDA2006#30 and CAIDA2007#25.

Since each algorithm is randomized and is executed 50 times to calculate the median error, standard deviations of output values are illustrated in Figure 5. These values are calculated for all algorithms at 6000 edges, accompanied by those in the edge-arrival model for the CAIDA-2006#30 dataset. This also confirms the reduction in variance by removing the unnecessary edges. Similar reductions will be observed at other space capacities, which we have excluded for brevity.

We conclude by presenting our results on the Reddit Hyperlinks dataset using oracles trained via machine learning. As described earlier, we construct edge features by combining features from the corresponding nodes. Due to missing node features, we are able to generate edge features for only 103K out of the 124K edges. We then train a logistic regression model to predict whether an edge participates in at least one triangle, and a linear regression model to estimate the triangle count of an edge (i.e., whether it is heavy). The first half of the edges is used as the training set for both models. The results of this approach are presented in Figure 6.

As we observe from figure 6, the proposed algorithm does not reduce the median errors in most cases. More importantly, as similarly illustrated in Chen's work [6], ThinkD outperforms their oracle-assisted approach and our algorithm as well. In the following section, we will discuss why this has happened.

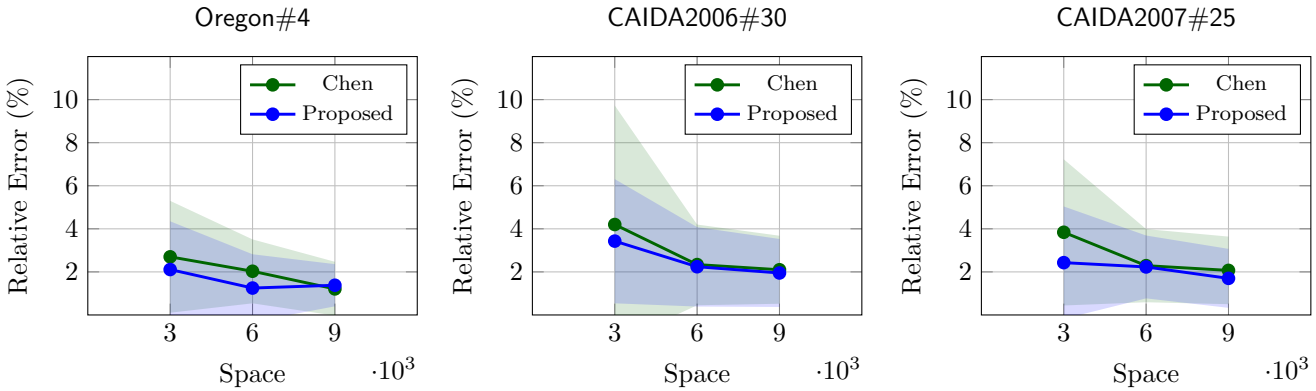

FIGURE 4. Comparison (vertex-arrival) OREGON and CAIDA datasets.

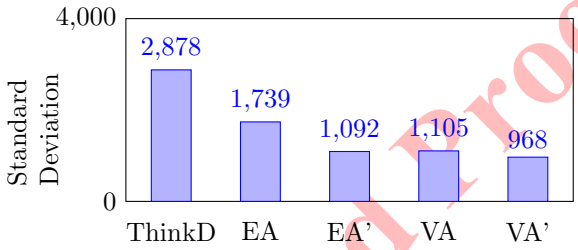

FIGURE 5. CAIDA2006#30, Standard deviation of outputs for Space = 6000 edges. (EA: Chen's Edge-Arrival Alg, VA: Chen's Vertex-Arrival Alg, ': removing unnecessary edges).

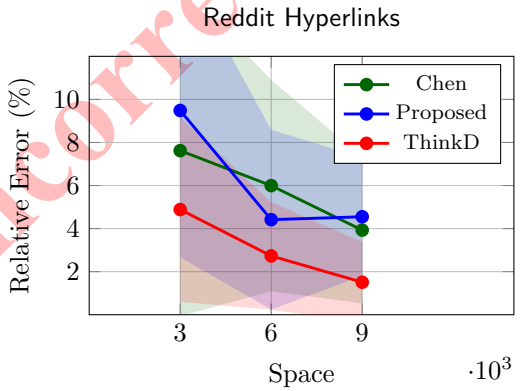

FIGURE 6. Comparison (ML-based oracles).

3.5. **Discussion.** In the previous section we showed how using an edge-triangle oracle can improve the accuracy of the learning-augmented algorithms in [6]. In practice, our main idea is to eliminate edges that do not participate in any triangle, thereby reducing unnecessary processing. While one might argue that identifying such edges requires additional space to track them, our intention is primarily to demonstrate that removing irrelevant edges can enhance the performance of state-of-the-art algorithms. In fact, oracles can be implemented as lightweight machine learning models that do not consume significant memory, yet can predict with high confidence whether an edge is likely to be

Table 5. Oracle accuracy (direct use of previous data and exact count).

Oracle	TP	TN	FP	FN	NI	Acc
OREGON-#1	11061	8534	522	517	2113	%94
CAIDA-2006-#1	16299	16836	1734	1589	9627	%90
CAIDA-2007-#1	19102	19906	1528	1540	9434	%92
Wiki-talk#1	26188	4319	2990	2781	1454299	%84

Table 6. Logistic regression accuracy.

Oracle	TP	TN	FP	FN	NI	Acc
Logistic Regression on Reddit Hyperlinks	81431	3148	16039	3147	20565	%81

part of a triangle. Such models are only trained on the first few instances of the graph and are subsequently saved and benefited from in the future instances to detect the unimportant edges. On datasets Oregon, CAIDA2006, and CAIDA2007, oracles are over 90% accurate and correctly remove over 8K unimportant edges (See table 5 for exact figures.) In this table, Positive means the oracle has declared an edge to be part of a triangle while Negative means the oracle has declared an edge unnecessary. NI means the oracle cannot make a prediction on that edge because it either had not appeared in previous instances or does not have a feature vector (representation) for the regression model.

However, our proposed algorithms do not generalize well across all datasets and may fail to produce accurate estimates, particularly in cases where the oracle-based approach of Chen et al. is itself ineffective (see Figures 3 and 6). We emphasize that the effectiveness of our method strongly depends on the oracle's ability to accurately identify edges that do not participate in any triangles. When we directly use previous data, the edge-triangle oracle, although not perfect, is accurate enough to help us outperform the algorithm of Chen et al. However, the accuracy of oracles that rely on the first $m/2$ (half of) edges are relatively lower. Specifically, when we incorporate learning into the problem, the accuracy drops to 81%. When we use logistic regression on the first half of the edges in Reddit Hyperlinks, our approach does not lead to a better estimate. This oracle only removes 3K unnecessary edges as well as mistakenly removing 3K important edges (See table 6 for exact figures.)

In fact, the non-oracle algorithm of ThinkD can outperform the oracle-assisted versions when oracles are less than 90% accurate. However, when the oracles become more accurate or perfect, the oracle-based algorithms (especially ours) outperform ThinkD. If we run our algorithm with a perfect oracle, our approach can easily outperform ThinkD. For instance, on Wiki-talk dataset and with a perfect oracle, our algorithm can remove 879K unnecessary edges and decrease the error to as low as 2% at the space of 15000 sampled edges. Thus, although such oracles can be built for some datasets (Oregon and CAIDA), the main challenge is building accurate or near-perfect oracles for any dataset of interest and generalize the approach.

4. Conclusion

In this paper, we addressed the problem of triangle counting in the streaming model under the assumption that the algorithm has access to oracles providing additional information about the input. Our focus was on the insertion-only setting, where edges are added but not deleted. We presented both theoretical bounds and supporting experimental results.

We proposed three algorithms in the arbitrary-order, insertion-only model. The first algorithm leverages an edge-triangle oracle, a novel oracle introduced in this work. The second algorithm assumes access to a degree oracle, while the third utilizes both oracles. The space complexities of the algorithms are $\tilde{O}(\epsilon^{-2}\Delta)$, $\tilde{O}(\epsilon^{-2}m^{3/2}/T)$, and $\tilde{O}(\epsilon^{-2}\sqrt{m})$ where the first algorithm works in a single pass while the others are 2-Pass algorithms.

In our experiments, we have implemented the idea of removing edges that do not participate in any triangles by calling oracles. These oracles can directly use previous instances of the graph, or they can be built by training machine

learning models. Our experiments show that in the autonomous systems' datasets, due to the high accuracy of edge-triangle oracles that directly use previous data, the proposed algorithms can lead to improvements over state-of-the-art (the algorithms of Chen et al. [6]). This happens specifically when space usage is low both in the edge-arrival and the vertex-arrival model.

However, on other datasets such as Reddit Hyperlinks and Wiki-talk, the oracles are not accurate enough to outperform even the non-oracle baseline of ThinkD. This observation highlights a key limitation of our approach: its effectiveness depends on the ability to construct accurate oracles. As part of future work, we plan to focus on improving the oracle component by exploring more powerful machine learning models—such as Graph Neural Networks—that can better capture structural patterns in graphs and make more reliable predictions across a broader range of datasets. Such oracles are particularly promising, as even large graphs with many triangles often contain a substantial number of edges that do not participate in any triangles.

References

[1] M. Al Hasan and V. S. Dave, *Triangle counting in large networks: a review*, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, *8*(2) (2018), e1226.

[2] N. Alon, R. Yuster, and U. Zwick, *Finding and Counting Given Length Cycles*, Algorithmica, *17*(3) (1997), 209–223.

[3] Z. Bar-Yossef, R. Kumar, and D. Sivakumar, *Reductions in streaming algorithms, with an application to counting triangles in graphs*, in SODA, *2* (2002), 623–632.

[4] S. K. Bera and A. Chakrabarti, *Towards tighter space bounds for counting triangles and other substructures in graph streams*, in 34th Symposium on Theoretical Aspects of Computer Science, 2017.

[5] A. Chakrabarti, *Data Stream Algorithms Lecture Notes*, 2020.

[6] J. Y. Chen, T. Eden, P. Indyk, H. Lin, S. Narayanan, R. Rubinfeld, S. Silwal, T. Wagner, D. P. Woodruff, and M. Zhang, *Triangle and four cycle counting with predictions in graph streams*, arXiv preprint arXiv:2203.09572, 2022.

[7] G. Cormode and H. Jowhari, *A second look at counting triangles in graph streams (corrected)*, Theoretical Computer Science, *683* (2017), 22–30.

[8] T. Eden, A. Levi, D. Ron, and C. Seshadhri, *Approximately Counting Triangles in Sublinear Time*, SIAM J. Comput., *46*(5) (2017), 1603–1646.

[9] H. Fichtenberger and P. Peng, *Approximately Counting Subgraphs in Data Streams*, in Proceedings of the 41st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, (2022), 413–425.

[10] R. Jayaram and J. Kallaugher, *An optimal algorithm for triangle counting in the stream*, arXiv preprint arXiv:2105.01785, 2021.

[11] H. Jowhari and M. Ghodsi, *New streaming algorithms for counting triangles in graphs*, in Computing and Combinatorics: 11th Annual International Conference, COCOON 2005, (2005), 710–716.

[12] N. Kavassery–Parakkat, K. M. Hanjani, and A. Pavan, *Improved triangle counting in graph streams: power of multi-sampling*, in 2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM), (2018), 33–40.

[13] S. Kumar, W. L. Hamilton, J. Leskovec, and D. Jurafsky, *Community interaction and conflict on the web*, in Proceedings of the 2018 World Wide Web Conference on World Wide Web, (2018), 933–943.

[14] J. Leskovec, D. Huttenlocher, and J. Kleinberg, *Governance in social media: A case study of the Wikipedia promotion process*, in Proceedings of the International AAAI Conference on Web and Social Media, *4*(1) (2010), 98–105.

[15] J. Leskovec, J. Kleinberg, and C. Faloutsos, *Graphs over time: densification laws, shrinking diameters and possible explanations*, in Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining, (2005), 177–187.

[16] A. McGregor, S. Vorotnikova, and H. T. Vu, *Better algorithms for counting triangles in data streams*, in Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, (2016), 401–411.

[17] R. A. Meyer, C. Musco, C. Musco, and D. P. Woodruff, *Hutch++: Optimal Stochastic Trace Estimation*, in 4th Symposium on Simplicity in Algorithms, SOSA 2021, (2021), 142–155.
[18] S. Muthukrishnan, *Theory of data stream computing: where to go*, in Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, (2011), 317–319.
[19] S. Muthukrishnan, *Data streams: Algorithms and applications*, Foundations and Trends in Theoretical Computer Science, *1*(2) (2005), 117–236.
[20] R. Pagh and C. E. Tsourakakis, *Colorful triangle counting and a mapreduce implementation*, Information Processing Letters, *112*(7) (2012), 277–281.
[21] A. Paranjape, A. R. Benson, and J. Leskovec, *Motifs in temporal networks*, in Proceedings of the tenth ACM international conference on web search and data mining, (2017), 601–610.
[22] A. Pavan, K. Tangwongsan, S. Tirthapura, and K.-L. Wu, *Counting and Sampling Triangles from a Graph Stream*, Proc. VLDB Endow., *6*(14) (2013), 1870–1881.
[23] C. Seshadhri, A. Pinar, and T. G. Kolda, *Fast triangle counting through wedge sampling*, in Proceedings of the SIAM Conference on Data Mining, *4* (2013), 5.
[24] K. Shin, *Wrs: Waiting room sampling for accurate triangle counting in real graph streams*, in 2017 IEEE International Conference on Data Mining (ICDM), (2017), 1087–1092.
[25] K. Shin, J. Kim, B. Hooi, and C. Faloutsos, *Think before you discard: Accurate triangle counting in graph streams with deletions*, in Joint European Conference on Machine Learning and Knowledge Discovery in Databases, (2018), 141–157.
[26] L. D. Stefani, A. Epasto, M. Riondato, and E. Upfal, *Triest: Counting local and global triangles in fully dynamic streams with fixed memory size*, ACM Transactions on Knowledge Discovery from Data (TKDD), *11*(4) (2017), 1–50.
[27] C. E. Tsourakakis, *Fast Counting of Triangles in Large Real Networks without Counting: Algorithms and Laws*, in Proceedings of the 8th IEEE International Conference on Data Mining (ICDM 2008), (2008), 608–617.
[28] J. S. Vitter, *Random sampling with a reservoir*, ACM Transactions on Mathematical Software (TOMS), *11*(1) (1985), 37–57.