Improved triangle counting in graph streams: Neighborhood multi-sampling

by

Kiana Mousavi Hanjani

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Science

Program of Study Committee:
Pavankumar Aduri, Major Professor
Samik Basu
Jia(Kevin) Liu

TThe student author, whose presentation of the scholarship herein was approved by the program of study committee, is solely responsible for the content of this thesis. The Graduate College will ensure this thesis is globally accessible and will not permit alterations after a degree is conferred.

Iowa State University

Ames, Iowa

2018

Copyright © Kiana Mousavi Hanjani, 2018. All rights reserved.

DEDICATION

I dedicate this thesis to my mother *Maryam Jamali Hanjani* along with my grandmother for their endless love, encouragement, support, and prayers from thousands of miles away. You believed in me when I did not believe in myself. I also want to thank my good friend *Alexey* for being so supportive and pushing me to my potential and reminding me of my capabilities.

Furthermore, I would like to dedicate this to all the young women and minorities in computing science who struggle every day to prove themselves, as well as all those who are living apart from their loved ones for the sake of science and education.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENTS

I would like to take this opportunity to express my thanks to those who helped me with various aspects of conducting research and the writing of this thesis. First and foremost, my advisor, Dr. Pavankumar Aduri, for his guidance throughout this research and the writing of this thesis. His insights and precision in proofs have often inspired me and renewed my passion for theoretical computer science.

I would also like to thank my committee members for their efforts and contributions to this work: Dr. Liu and Dr. Basu.

I would like to thank Dr. Basu once more for being such a caring and patient *Director of Graduate Education* in our department. Without his mentorship and words of wisdom, I would have been lost.

I would like to thank the Chair of our department Dr. Gianfranco Ciardo for his attempts to make our time here more fruitful as well as our Program Coordinator Mrs. Carla Harris for her constant support and patience with the students.

ABSTRACT

In this thesis, we study the problem of estimating the number of triangles of an undirected graph in the data stream model. Some of the well-known streaming algorithms work as follows: Sample a single triangle with high enough probability and repeat this basic step to obtain a global triangle count. For example, the neighborhood sampling algorithm attempts to sample a triangle by randomly choosing a single edge e, a single neighbor f of e and waits for a third edge that completes the triangle. The basic sampling step in the algorithm is repeated multiple times to obtain an estimate for the global triangle count in the input graph stream. In this work, we propose a multi-sampling variant of this algorithm. We provide a theoretical analysis of the algorithm and prove that it improves upon the known space and accuracy bounds. We experimentally show that this algorithm outperforms several well-known triangle counting streaming algorithms.

CHAPTER 1. INTRODUCTION

The intelligent data analysis has passed through a number of stages. Each stage addresses novel research issues that have arisen. Due to the increase in database sizes, new algorithms have been proposed to deal with the scalability issue. The recent advances in hardware and software have enabled the capture of different measurements of data in a wide range of fields. These measurements are generated continuously and in a very high fluctuating data rates. Examples include sensor networks, weblogs, and computer network traffic. The storage, querying and mining of such data sets are highly computationally challenging tasks [11]. Recently, the data generation rates in some data sources become faster than ever before. This rapid generation of continuous streams of information has challenged our storage, computation and communication capabilities in computing systems. Systems, models, and techniques have been proposed and developed over the past few years to address these challenges [1, 23].

1.1 Data Streaming Model

A data stream is a sequence of data elements made available through time. In the data stream model, some or all of the input is represented as a finite sequence of element(from some finite domain) which is generally not available for random access, but instead arrives one at a time in a stream. If the stream has length n and the domain has size m, algorithms are generally constrained to use space that is logarithmic in m and n.

Often it is the case that the stream processor does not have enough resources(space) to store the entire stream in its memory and perform an off-line computation. Thus the processor cannot access an individual item of the stream multiple times unless it is stored in the memory. Hence streaming algorithm needs to run in real time, as the data arrives and decide if each element will be processed (in our case, sampled) or discarded. Each time the algorithm runs through the stream is called a *pass* over the data. Streaming algorithms' performances are usually compared by the number of passes, available memory and their runtime. Since exact answers are not possible due to limited memory, we settle for an approximate answer. *Approximation algorithms* can result in an approximate solution with error bounds. Sampling and sketching [23, 1] techniques are common techniques used in this case. Sampling refers to the process of probabilistic choice of a data item to be processed or not. Boundaries of the error rate of the computation are given as a function of the sampling rate.

The data streams we study here are graph streams, meaning they correlate with the changes in a graph network with time. The edges arrive as a stream in time. Each edge corresponds to a relation between two vertices of the graph. Sampling such streams is the act of choosing a subgraph of the main graph and process it instead.

If the stream is not dynamic, the arriving elements of the stream are new edges being added to the graph network. But in a dynamic graph stream, edges can be added or removed, in case two vertices are no longer connected. So the elements will consist of the edge and the operation (add or delete). Studying dynamic streams are much harder since the sampling algorithm needs to be considering the deletions as well.

1.2 Triangle substructure

Graphs are data structures that used to model complex relations in a wide variety of applications, including biochemistry, neurobiology, ecology, social sciences, and information systems. Normally, the graph consists of a set of nodes or vertices and a second set of edges which are the relations between those vertices. These relations can be one-sided(directed graphs) or two-sided(undirected graphs). With the growth of networks, the demand for their graph representations has also increased. Defining new measures of interest on graph data and designing novel algorithms that compute or approximate such measures on large graphs is an important task for analyzing graph structures that reveal their underlying properties [4].

A triangle in an undirected simple graph is a subgraph of three nodes that are all connected. A triangle is also the smallest non-trivial clique. In an $n-$clique, there are exactly $\binom{n}{3}$ triangles.

Understanding coarser and finer properties of triangle structures in a network has found applications in social network analysis, recommendation systems, spam and fraud detection, and understanding structure and evolution of social and web graphs [35, 5, 10, 12, 26, 3, 9, 33]. Sociologists have long been interested in triadic structures in social network graphs [35, 5]. The frequently used notions of *transitivity coefficient* and *clustering coefficient* critically depend on the number of triangles present in a social network. With the advent and presence of very large scale graphs such as online social networks and web graphs, the computational complexity of computing the number of triangles in a graph has received a lot of attention. Since exact triangle computation is expensive on large-scale graphs, considerable emphasis has been placed on designing algorithms that will *estimate* number of triangles of a graph with *provable guarantees* on the quality of the estimation. The problem we are trying to contribute to here is counting the number of triangles in a streaming graph in one pass. There have been several algorithms that count the number of triangles in a data stream, with different approximations, run times and memory bounds. We will show our contribution in the following chapters starting with literature review, details about our algorithm, proof that the algorithm works, our memory bound and end with experimental results comparing our algorithm to the previous ones. We will see that our algorithm gives better memory bounds in theory and also in practice.

CHAPTER 2. PRIOR WORK

There has been an extensive body of research on the problem of exact and approximate triangle counts in various computational models [27, 31, 33, 32, 34, 30, 24, 18]. For example in [18], the authors study triangle counting in very large sparse graphs and try to focus on the space complexities of known algorithms and discuss their implications. In [27], they evaluate the practicability of triangle counting and listing in very large graphs with various degree distributions. Tsourakakis et al. propose a practical method in [32] using a straightforward triangle counting algorithm as a black box and performing experiments on real-world networks and on synthetic data sets.

Here we restrict our attention to the prior in the data stream model. Bar-Yosseff *et al.* were the first to formally study the problem of triangle counting the data stream model in 2002 [2]. They reduced the triangle counting problem on graphs to estimating zeroth and second frequency moments over numeric data. Much of the latter work is based on various sampling strategies in the sense that they attempt to sample a sub-graph of the original graph [21, 16, 6, 28, 13, 14, 15, 8, 25, 20]. Some of these works use reservoir sampling to sample edges/vertices, whereas other use fixed probability to sample edges/triangles. Reservoir sampling is the family of randomized sampling algorithms used for randomly choosing a sample of k items from a list S containing n items, where n is either a very large and does not fit in memory or is an unknown number. The idea is to keep at most k items in memory at all times, deleting the previous items either by timestamp or randomly to update the sample. As for fixed probability algorithms, the work of Jha et al. in [13] is based on the classic probabilistic result, the birthday paradox. They prove that when the transitivity is constant and there are more edges than vertices (common properties for social networks), their algorithm requires $O(\sqrt{n})$ space to require accurate results. The work of Lim and Kang [20] uses a fixed probability to sample the edges. They fix a parameter p and pick each edge with probability p and count the number of triangles formed by the sampled sub-graph. As for the reservoir sampling algorithm, the most recent example is the Triest based algorithm [29] which we will discuss thoroughly later in this section.

Some of the prior work on *insert-only streams* (i.e. edge deletions are not allowed) [13, 14, 15, 8, 25], whereas other algorithms work on dynamic graph streams that allow deletion of edges [7, 29]. Some algorithms provide the triangle count only after the entire stream is processed, while other algorithms can continuously monitor the number of triangles in the graph that has been observed so far.

The memory used by these algorithms depend on two factors: Various graph parameters such as number of vertices (n), number of edges (m), total number of triangles (T), maximum-degree (Δ), etc, and the approximation and error probability factors ϵ and δ. The memory bounds of various algorithms can be found in Table 3.1 in the next section.

The algorithm due to Pagh and Tsourakakis[24], to which we refer to as *colorful triangle counting*, each vertex of the graph is colored with $N = 1/p$ colors randomly. The triangles which have the same colored vertices are then counted. They claim that when $p \geq max(\frac{\delta logn}{t}, \sqrt{\frac{logn}{t}})$, the triangle estimate is concentrated around its expectation.

The streaming algorithm of Buriol *et al.* [8]works as follows: uniformly at random, using the well-known reservoir sampling technique, pick a vertex v and an edge $\langle a, b\rangle$ of the graph. Once sampled, if the both *cross edges* $\langle a, v\rangle$ and $\langle b, v\rangle$ arrive in the stream, then output 1 otherwise output 0. Run R *independent* copies of this basic procedure and take the average output as an estimate for the number of triangles. They have proved that the output is a (ϵ, δ) approximation of the total number of triangles when the space/memory used is $O(\frac{mn}{T}\frac{1}{\epsilon^2}\log(1/\delta))$. Here T denotes the total number of triangles in the graph. From now on, we will refer to this algorithm as *Edge-Vertex Single Sampling algorithm*, EVSS algorithm for short.

The neighborhood sampling algorithm [25] attempts to sample a triangle by randomly choosing a single vertex v, a single neighbor u of v and waits for a third edge that completes the triangle. This algorithm uses reservoir sampling to uniformly at random pick an edge e_1, a random neighboring edge e_2 of e_1, and waits for an edge e_3 such that e_1, e_2, e_3 form a triangle. It is shown that this algorithm uses $O(\frac{m\Delta}{T}\frac{1}{\epsilon^2}\log(1/\delta))$ to compute a (ϵ, δ) approximation of T. Here Δ denotes the maximum degree of the graph. The authors refer to this algorithm as *neighborhood sampling algorithm.*

Both of the above-described algorithms attempt to sample a *single triangle.* In our previous work [17], by modifying the above procedures to sample multiple triangles, we obtain much better theoretical bounds on the memory and space used. Consider EVSS algorithm. Instead of picking a single vertex and a single edge, sample *multiple vertices and multiple edges* , store the *cross edges* that connect the sampled vertices to the sampled edges and count the number of triangles formed by sampled edges and cross edges and scale the number by an appropriate factor. This is referred to as the EVMS algorithm [17, 22].

The recent work of Stefani *et al.*[29], called the *Triest* algorithm, uses reservoir sampling to sample multiple edges and counts the number of triangles in the sampled-subgraph. Their algorithm also handles dynamic streams (where edges can be deleted) and can be used to obtain local-triangle counts.

In this work, we propose an extension to the *neighborhood sampling* algorithm called the *neighborhood multi-sampling* algorithm, NMS for short, using the idea of *multi-sampling.* We will prove that our algorithm gives better memory bounds for streaming graphs in theory as well as in the experiment over several different data-sets. It is also faster and more consistent than some of the previous well-known algorithms.

CHAPTER 3. ALGORITHMS AND ANALYSIS

In this section, we present the pseudo code and explanation of the NMS algorithm as well as its accuracy and memory bounds. We start with some preliminaries and then go into details of the algorithm analysis to gain the memory bounds.

3.1 Neighborhood Multi-Sampling algorithm(NMS Algorithm)

We assume the graph stream, G_S arrives element by element in the form of $e_i = (u, v)$, in which e_i is an edge with two end points u and v. fix two probability parameters p and q. The algorithm maintains three sets of edges L_1, L_2, and L_3. When an edge e arrives it is placed in L_1 with probability p. In addition, if e is a neighbor of an edge in L_1, then it is placed in L_2 with probability q. Moreover, if e forms a triangle with an edge from L_1 and L_2, then it is placed in L_3. When triangle estimate is needed, we compute all triangles with edges e_1, e_2, e_3 of the form: $e_1 \in L_1$, $e_2 \in L_2$, $e_3 \in L_3$ and e_1 arrived before e_2, and e_2 arrived before e_3. We scale the count by dividing the number of triangles by pq and this is our estimate for the total number of triangles in the graph. Though this description of the algorithm uses L_3, we can implement the algorithm without explicitly storing L_3. The pseudo-code of the algorithm is shown in Figure 3.1.

3.1.1 Preliminaries

We briefly describe the notations/ theorems that are used in the paper.

We say two triangles are incident on a vertex v or edge e if they both contain v as one of their vertices or e as one of their edges. For a graph $G = (V, E)$, we use n and m to denote the number of vertices and edges respectively. We use T to denote the total number of triangles, T_v to denote the number of triangles incident on a vertex v, T_e to denote the number of triangles incident on an edge e, Δ to denote the maximum degree of the graph. The maximum number of triangles a

```
procedure NMS Algorithm
    Input: The graph stream $G_S$
    Initialize: $L_1 = \emptyset$, $L_2 = \emptyset$, $L_3 = \emptyset$, $Y = \emptyset$
    for each edge $e_i = (u, v)$ in $G_S$, do
        With probability $p$, add $e_i$ to $L_1$
        if $e \in N(L_1)$ then
            With probability $q$ add $e_i$ to $L_2$
        end if
        for every $(e_j, e_k)$, $e_j \in L_1$, $e_k \in L_2$ such that $t(e_j) < t(e_k)$ and $t(e_j) < t(e_i)$ and $\langle e_i, e_j, e_k \rangle$ form a triangle do
            Add the triangle $\langle e_i, e_j, e_k \rangle$ to $Y$
        end for
    end for
    Output: $|Y|/pq$, Number of sampled triangles from the stream
end procedure
```

Figure 3.1: NMS Algorithm

vertex or an edge can participate in are denoted by Δ_V and Δ_E. Note that $\Delta_E \leq \Delta_V \leq \Delta^2$.

Defintion. We say that a probabilistic algorithm computes an (ϵ, δ) approximation of T if the output of the algorithm lies between $T + \epsilon T$ and $T - \epsilon T$ with probability $\geq 1 - \delta$.

Chebyshev's Inequality: Let X be a random variable. Then,

$$Pr[|X - E(X)| \geq \delta] \leq \frac{Var(X)}{\delta^2}$$

Chernoff Inequality: Let $X_1, X_2, ..., X_m$ be independent random variables that take values between 0 and 1. Let $E(X_1) = E(X_2) = ... = E(X_m) = p$. Let $X = X_1 + X_2 + ... + X_m$, then

$$Pr[|X/m - p| \geq p\delta] \leq 2e^{\frac{-\delta^2 mp}{2}}$$

Hajnal-Szemeredi Theorem: Every graph with n vertices and maximum vertex degree at most k is $k+1$ colorable with all color classes of size at least $\frac{n}{k+1}$.

3.1.2 Analysis

Let $\tau(G)$ be the set of all the triangles in the graph G and $|\tau(G)| = T$.

Lemma 3.1.1 *Let $t \in \tau(G)$. The probability that t is sampled in the* NMS *algorithm is, $Pr[t \in Y] = pq$.*

Proof. A triangle is sampled if the L_1 edge is sampled with probability p, and then the L_2 edge in the neighborhood of the L_1 edge is sampled with probability q. Since every edge is sampled with a fixed probability, p and q respectively, and the two events are independent, the probability that a triangle, $t \in \tau(G)$ gets sampled is pq.

Lemma 3.1.2 *The* NMS *algorithm outputs X whose expectation is the number of triangles in the graph G*

Proof. Let's define random variables $X_1, X_2, X_3, \ldots, X_T$, such that,

$$X_i = \begin{cases} 1 & \text{if the } i^{th} \text{ triangle of } G \text{ gets sampled} \\ 0 & \text{otherwise} \end{cases}$$

Let p be the probability of the i^{th} triangle to be sampled according to lemma 3.1.1 is:

$$Pr[X_i = 1] = pq$$

Then the expected value of X_i, would be:

$$E[X_i] = Pr[X_i = 1].1 + Pr[X_i = 0].0 = pq$$

Let us define a new random variable X as

$$X = X_1 + X_2 + ... + X_T$$

The expected value of X will be

$$E[X] = E[\sum_{i=1}^{T} X_i] = \sum_{i=1}^{T} E[X_i] = Tpq$$

.

Recall that

$$Var[X] = \sum_{i=1}^{T} Var(X_i) + \sum_{0 \leq i \leq j \leq T} Cov[X_i, X_j]$$

From the definition of Variance for Bernoulli random variables, $Var[X_i] = pq - p^2q^2 \leq pq$. Note that if triangles i and j do not share an edge then the incident of them being sampled is completely independent and $Cov[X_i, X_j] = 0$. Triangles i and j can be dependent when they share an edge. If they share a vertex then $E[X_iX_j] = pq^2$. We want get an upper bound for the variance. So we try to get a bound for $Cov(X_i, X_j)$. Since $Cov(X_i, X_j) = E[X_iX_j] - E[X_i]E[X_j]$ which is at most $E[X_iX_j]$. Thus $Cov(X_i, X_j) \leq pq^2$. However, if the triangles share an edge, then $E[X_iX_j] \leq pq^2$. In this case the Co-variance of X_i and X_j is bounded by pq^2. Let P denote the maximum of p and q. So if we sum all such covariances, we have

$$\begin{aligned}\sum_{0 \leq i \leq j \leq T} Cov(X_i, X_j) \quad & \leq Ppq(\textstyle\sum_{e \in G} T_e^2) \\ & \leq Ppq(\textstyle\sum_{e \in G} T_e \Delta_E) \\ & \leq Ppq\Delta_E \textstyle\sum_{e \in G} T_e \\ & \leq 3Ppq\Delta_E T\end{aligned}$$

Here T_e is the number of triangles incident on an edge e and $\Delta_E = \max_e T_e$. Thus $Var(X)$ is bounded by $Tpq(1 + 3P\Delta_E)$. Using Chebyshev inequality, we can obtain a concentration bound for X. Recall that the output of the algorithm is $\dfrac{X}{pq}$. So if the variance for $\dfrac{X}{pq}$ is σ^2 we will have:

$$Pr[|\frac{X}{pq} - T| \geq \epsilon T] \leq \frac{\sigma^2}{\epsilon^2 T^2}$$

$\sigma^2 = \dfrac{Var[X]}{pq}$ and we already have an upper bound for $Var[X]$.

$$Pr[|\tfrac{X}{pq} - T| \geq \epsilon T] \leq \tfrac{Tpq(1+3P\Delta_E)}{p^2q^2\epsilon^2T^2} \leq \delta$$

So to obtain an ϵ, δ approximation we require that

$$pq \geq \frac{1 + 3P\Delta_E}{\delta\epsilon^2}$$

For simplicity, if set $p = q$, after solving this inequality we get $q \geq \frac{-3\Delta_E + \sqrt{q\Delta_E^2 + 4\epsilon^2\delta}}{2\delta\epsilon^2}$. So the bound given by Chebyshev's inequality is

$$pq = p^2 \geq \frac{-3\Delta_E + \sqrt{q\Delta_E^2 + 4\epsilon^2\delta}}{4\delta^2\epsilon^4}$$

Thus the above algorithm outputs (ϵ, δ) approximation if $p = q$ and the above inequality holds.

However, we now show how to achieve a better concentration bound using Hajnal-Szemeredi theorem and Chernoff bounds. Note that the random variables $X_1 \cdots X_T$ are not mutually independent thus Chernoff bounds cannot be applied directly. This problem can be gotten around by appealing to the work of Hajnal-Szemeredi. Pagh and Tsourakakis are the first to apply this technique in the context of triangle counting [24].Let X_i be defined as above, $i = 1, ..., T$. Consider an auxiliary graph H. For every X_i (equivalently a triangle t_i) place a vertex v_i in H. If X_i and X_j are dependent, place an edge between v_i and v_j. as follows(shown in Figure 3.2).

Add a vertex in H for every triangle in G. Then connect two vertices v_1 and v_2 in H representing triangles t_1 and t_2 in G, if and only if t_1 and t_2 have a common edge.

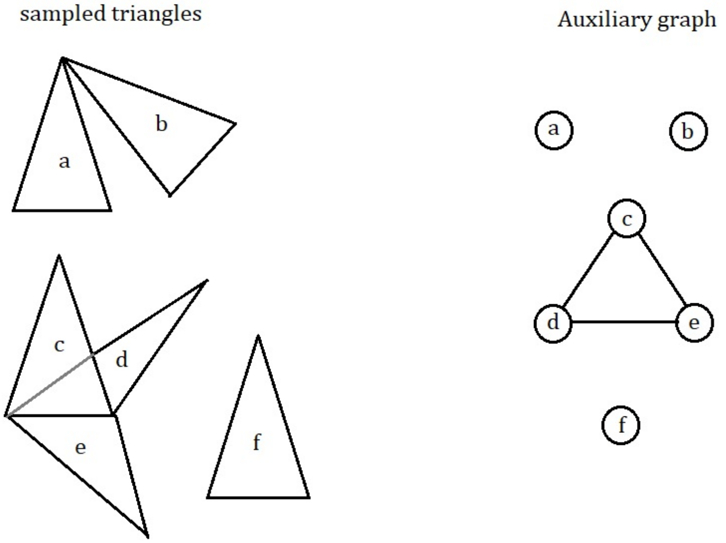

Figure 3.2: Construction of auxiliary graph from the sampled triangles

Observe that the maximum degree of H is Δ_E. According to Hajnal-Szemeredi theorem, the graph G can be colored with $\Delta_E + 1$ colors, with each color set having at most $k = \dfrac{T}{\Delta_E}$ vertices. Consider such a coloring. Note that if v_1 and v_2 have the same color, the corresponding random variables X_i and X_j are independent. Thus all the random variables belonging to same color class are mutually independent and we can apply Chernoff bounds in each such class. For a color c, let k_c denote the number of vertices of H with color c. Note that T equals the sum of all k_c's. Fix a color c, and without loss of generality, assume that $X_1, \cdots, X_{k_c}$ $t_1, t_2, \cdots t_{k_c}$ be the random variables corresponding to the color class c. Let $Z_c = \sum_{i=1}^{k_c} \frac{X_i}{pq}$. Note that $E[Z_c] = k_c$. By Chernoff bounds,

$$\begin{aligned} Pr[|Z_c - k_c| \geq \epsilon k_c] &= Pr[|\frac{\sum_{i=1}^{k} X_i}{p_v p_e} - k_c| \geq \epsilon k_c] && (3.1) \\ &= Pr[|\frac{\sum_{i=1}^{k} X_i}{k_c} - pq| \geq \epsilon pq] && (3.2) \\ &\leq 2e^{\frac{-\epsilon^2 pqk}{2}} && (3.3) \end{aligned}$$

Assume $C = \{c_1, c_2, ..., c_r\}$ are our colors. Define $Z = \sum_{c \in C} Z_c$. Note that the output of the algorithm is the random variable Z. Call a color c *good* if $|Z_c - k_c| \leq \epsilon k_c$. If all colors are good, then

$$\begin{aligned} \sum_{c \in C} k_c - \sum_{c \in C} \epsilon k_c \leq \sum_{c \in C} Z_c &\leq \sum_{c \in C} k_c + \sum \epsilon k_c \\ T - \epsilon T \leq Z &\leq T + \epsilon T \end{aligned}$$

Thus if all colors are good, then the output of the algorithm (Z) differs from T by at most ϵT. By Inequality 3.3, the probability that a color class is not good is at most $2e^{\frac{-\epsilon^2 pq}{2}}$. Since there are at most $\Delta_E + 1$ many color classes, by union bound, the probability that some color class is not good is $(\Delta_E + 1)(2e^{\frac{-\epsilon^2 pqk}{2}})$. If we want this probability to be less than δ, then we obtain that

$$pq \geq O(\frac{\Delta_E}{\epsilon^2 T} log \frac{\Delta_E}{\delta})$$

Thus when the above inequality holds, our algorithm outputs an (ϵ, δ) approximation to T.

3.1.3 Memory bound

We now analyze the memory used by the algorithm. The memory used for this algorithm consists of the memory used to store the L_1, L_2 edges. The expected number of L_1 edges in total is pM. We can compute the expected number of L_2 edges as follows: $(1-(1-p)^{t_e})q$ in which $t(e)$ is the number of neighbors of e coming before e in the stream. So the total memory of second level edges will be:

$$\textstyle\sum_{e\in G}[1-(1-p)^{t(e)}]q$$

$$= q\textstyle\sum_e 1 - \textstyle\sum_e (1-p)^{t(e)}$$

$$= q(m - \textstyle\sum_e (1-p)^{t(e)}) \leq qm$$

So the memory can be simplified to $pm + qm$ such that $pq \geq \frac{\Delta_E}{\epsilon^2 T} log \frac{\Delta_E}{\delta}$

Theorem 3.1.3 *There is a streaming algorithm that computes (ϵ, δ)-approximation of number of triangles in a graph by using expected space $O(mp)$ where p is $O(\sqrt{\frac{\Delta_E}{T}} \frac{1}{\epsilon} \sqrt{log \frac{\Delta_E}{\delta}})$.*

Memory of a triangle counting algorithm can be written as $f(G).g(\epsilon, \delta)$ in which f is a function of the graph and g is a function of ϵ and δ. In the table below, you can see f and g for some of the recent algorithms versus the proposed algorithms in this paper. The algorithms referred to are Edge Vertex Simple Sampling algorithm (EVSS)by Buriol *et al.* [8], Neighborhood Sampling by Pavan et al's [25], Triest-Base algorithm [29], Colorful Triangle Counting by Pagh *et al.*'s [24], EVMS algorithm [17] and the NMS algorithm.

Table 3.1: Comparison of the algorithms

	EVSS	NS	Triest	Colorful	EVMS	NMS
$f(G)$	$\frac{mn}{T}$	$\frac{m\Delta}{T}$	$m(\frac{\Delta_E}{T})^{\frac{1}{3}}$	$m\sqrt{\frac{\Delta_V}{T}}$	$m\sqrt{\frac{\Delta_V}{T}}$	$m\sqrt{\frac{\Delta_E}{T}}$
$g(\epsilon, \delta)$	$\frac{1}{\epsilon^2} log \frac{1}{\delta}$	$\frac{1}{\epsilon^2} log \frac{1}{\delta}$	$\frac{1}{(\epsilon)^{\frac{2}{3}}} \log^{2/3} \frac{\Delta_E}{\delta}$	$\frac{1}{\epsilon} \log \frac{1}{\delta}$	$\frac{1}{\epsilon}\sqrt{log \frac{\Delta_V}{\delta}}$	$\frac{1}{\epsilon}\sqrt{log \frac{\Delta_E}{\delta}}$

CHAPTER 4. EXPERIMENTAL RESULTS

4.1 Experimental Data

The experiments were performed to analyze the memory usage, accuracy, and speed of both the algorithms, and to empirically verify the performance of the algorithms for varying values of p and q. We ran the algorithms with the probabilities set to $0.01, 0.05, 0.1, 0.15$ and 0.2. Hence for each graph, we have 25 possible experiments for each algorithm. The same experiment were done for the EVMS algorithm in [22].

Experiments were conducted on several graphs from the Stanford Large Network Dataset Collection [19]. The properties of these datasets are shown in Table 4.1.

Amazon product co-purchasing network: Network was collected by crawling Amazon website. It is based on *Customers Who Bought This Item Also Bought* feature of the Amazon website. If a product i is frequently co-purchased with product j, the graph contains an undirected edge from i to j.

Skitter: An Internet topology graph from trace routes run daily in 2005 from several scattered sources to million destinations.

LiveJournal social network: A free on-line blogging community where users declare friendship with each other.

DBLP Collaboration network : A co-authorship network of computer science bibliography where two authors are connected if they publish at least one paper together. Authors who published to a certain journal or conference form a community.

Orkut social network: A free on-line social network where users form friendship each other. Orkut also allows users form a group which other members can then join.

Berkeley-Stanford web graph: Nodes represent pages from berekely.edu and stanford.edu domains and edges represent hyperlinks between them. The graph was initially directed, which was

converted to undirected by removing redundant edges that represent direction.
The implementation was done in Java 8, and run on a PC with Core i7 2.5 Ghz, 16GB RAM with Windows 10 operating system. The experiments were run on insertion only streams for the NMS algorithm, where each edge is a new edge with no repetitions. The datasets were chosen in such a way that they have varying topologies. The graphs have varying densities ranging from around 900,000 edges (Amazon) to more than 100 Million edges (Orkut).

All the graphs were pre-processed so that the vertices are numbered from 0 to $n-1$, where n is the total number of vertices of the graph.

4.2 Experiments for the NMS algorithm

We compare the performances of our algorithms against EVSS algorithm from Buriol *et al*'s [8], Triest-Base [29], neighborhood sampling algorithm [25], colorful sampling algorithm due to Pagh and Tsourakakis [24] and the EVMS algorithm [17]. The rationale for this choice of algorithms is as follows: Naturally, we would like to compare NMS against itscounter parts. In [29], the authors experimentally showed that Triest-Base algorithm outperforms several other algorithms from the literature, thus we chose Triest-Base algorithm. The colorful triangle counting algorithm was chosen because as far as our knowledge its performance on graph streams has not been tested before. We re-implemented these algorithms in java. The theoretical memory bounds for the these algorithms are shown in 3.1. We calculated memory usage of the algorithms as the total number of edges sampled. We first ran the NMS algorithm for various choices of p and q. To get a fair comparison, the other algorithms were run using (approximately) same amount of total memory as used by the NMS algorithm. We measured the accuracy of the algorithms by using percentage error, $pE = 100(T-\tau)/T$, where T is the actual number of triangles in the graph and τ is the estimated triangle count by an algorithm. The total memory is the sum of the level 1 and level 2 edges for the NMS algorithm.

Once sampled, we used edge-iterator to count the number of triangles among the sampled edges. The NMS algorithm shows high accuracy even when $p = q = 0.01$. For all experiments, it is

Table 4.1: Properties of the various datasets considered for the experiments

Graph	Nodes	Edges	Triangle count
DBLP	317,080	1,049,866	2,224,385
As-Skitter	1,696,415	11,095,298	28,769,868
Live Journal	3,997,962	34,681,189	177,820,130
Amazon	334,863	925,872	667,129
Berk-Stanford	685,230	7,600,595	64,690,980
Orkut	3,072,441	117,185,083	627,584,181

observed that the total memory used is strictly less than the theoretical upper bound of $m(p+q)$. The results are summarized in Table 4.2.

4.2.1 NMS vs EVMS Sampling

The comparison between the two new algorithms proposed in this work are in Table 4.3. We can observe that the total memories used by both the algorithms are very close to each other when $p = p_e$ and $q = p_v$, with the NMS using slightly lesser memory than the EVMS algorithm. Figure 4.8 plots the total memory T_m against various sampling probabilities ($p_v = q$ and $p_e = p = 0.05$) on the Skitter graph. In terms of accuracy, we see that NMS algorithm is far superior on all the experimental settings–nearly three to six times more accurate than the EVMS algorithm. From the memory vs q plot in Figure 4.8, the total memory used increases almost linearly with the increase in probability, whereas in EVMS, the memory increase is not perfectly linear. This partly explains the high variance in the estimate from the EVMS algorithm, where the pE sometimes increases even though T_m increases, as plotted in Figure 4.4.

In terms of the time taken, the NMS algorithm is slower mainly due to the fact that computing number triangles once the edges are sampled can be done more efficiently in EVMS algorithm by maintaining efficient data structures. It can be estimated that if R is the number of edges sampled, then EVMS triangle count takes $O(R)$ time, where as NMS triangle count takes $O(R \log R)$ time. Additional overhead involved in checking for time stamps involving all three edges of a triangle. In the EVMS technique, for every red edge, we find all neighboring black edge with common vertices such that every neighboring edge has a time stamp greater than the red edge. This can be done in $O(|N|)$ time, where N is the set of all neighbors for a red edge. In the NMS technique for every level 1 edge, e, we find all common neighbors n_1 and n_2 from the neighbor set N of e, such that they share a vertex. This operation takes $O(|N|^2)$. This explains the higher running time of the NMS algorithm.

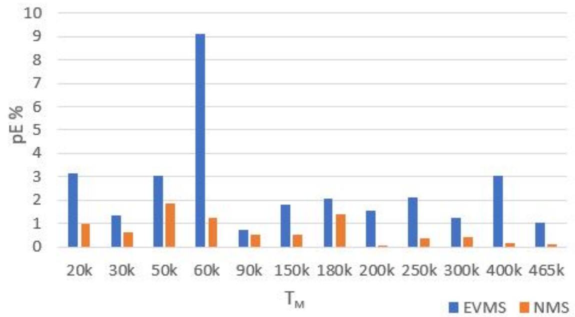

Figure 4.2 DBLP

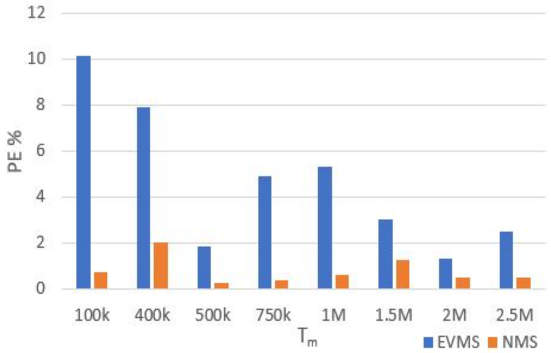

Figure 4.3 Berkeley Stanford

Figure 4.4: T_m vs pE for NMS and EVMS algorithms for DBLP

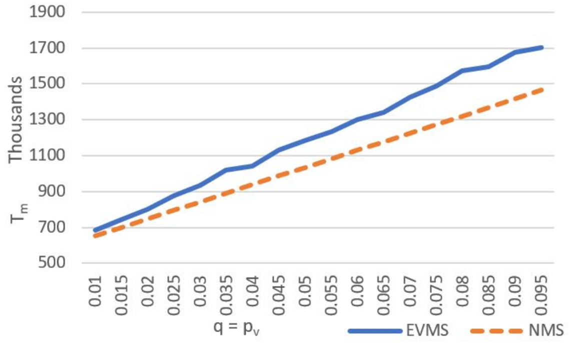

Figure 4.6 As-Skitter

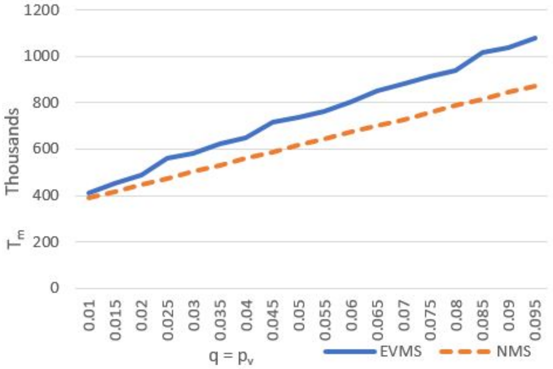

Figure 4.7 Berkeley Stanford

Figure 4.8: T_m vs $p_v = q$ when $p_e = p = 0.05$ for NMS and EVMS algorithms for As-Skitter graph

4.2.2 NMS vs Triest based

The detailed comparison results are shown in table 4.4. The experiments show that the NMS algorithm is much faster than the Triest base, in both small and large graphs. the error for the NMS algorithm is consistently low although for the Triest, the error changes radically and increases as the size of the graph gets larger.

4.2.3 NMS vs Colorful triangle counting

The detailed comparison results are shown in Table 4.5. Though the NMS is slower than the EVMS algorithm, it is still faster when compared to colorful triangle counting for small graphs. For larger graphs the colorful triangle counting does better than NMS in terms of time. The time taken by the algorithms for the Berkeley-Stanford graph is shown in Figure 4.12. The NMS algorithm shows better accuracy in most of the cases.

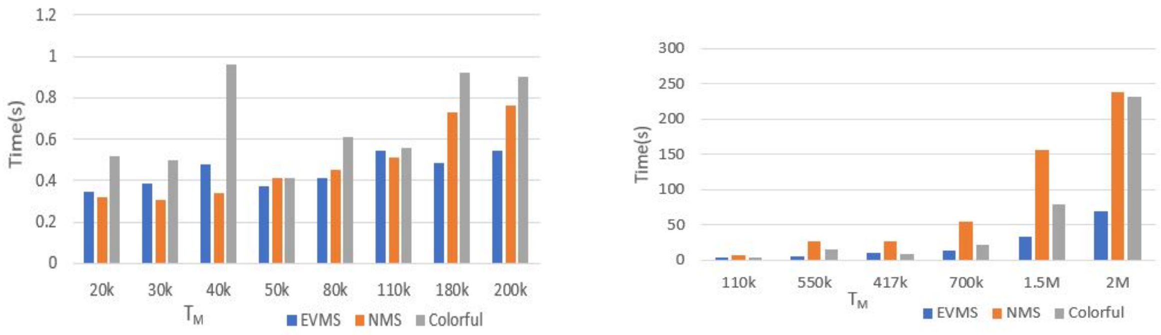

Figure 4.10 DBLP

Figure 4.11 Berkeley Stanford

Figure 4.12: T_m vs $Time$ for EVMS, NMS and Colorful triangle counting algorithms for Berkeley Stanford graph

Table 4.2: Results of the NMS algorithm for various graphs

As-skitter

p	q	T_m	τ_{neigh}	pE	Time (s)
0.01	0.01	179,235	27,320,000.00	-5.04	6.22 3
0.05	0.15	1,989,823	28,307,333.33	-1.61	51.46
0.10	0.15	2,664,100	28,724,533.33	-0.16	84.77

Live-journal

p	q	T_m	τ_{neigh}	pE	Time (s)
0.01	0.01	513,674	175,570,000.00	-1.2654	14.76
0.05	0.20	7,656,244	177,739,500.00	-0.0453	60.15
0.10	0.15	8,379,284	177,826,600.00	0.0036	74.37

Orkut-social network

p	q	T_m	τ_{neigh}	pE	Time (s)
0.05	0.01	6,999,753	625,484,000.00	-0.33	135.13
0.10	0.01	12,888,478	628,859,000.00	0.20	189.67
0.15	0.01	18,746,995	627,811,333.33	0.04	273.70

DBLP social network

p	q	T_m	τ_{neigh}	pE	Time (s)
0.01	0.05	19,764	2,202,000.00	-1.01	0.32
0.15	0.01	166,237	2,226,666.67	0.10	0.54
0.10	0.10	181,344	2,223,600.00	-0.04	0.76

Amazon Product Graph

p	q	T_m	τ_{neigh}	pE	Time (s)
0.01	0.05	13,744	656,000.00	-1.67	0.35
0.10	0.05	118,501	671,200.00	0.61	0.67
0.15	0.01	145,885	668,666.67	0.23	0.56

Berkeley-Stanford web graph

p	q	T_m	τ_{neigh}	pE	Time (s)
0.01	0.05	283,825	60,962,000.00	-5.7643	24.11
0.10	0.10	1,276,129	63,879,600.00	-1.2542	186.67
0.20	0.05	1,649,959	64,368,500.00	-0.4985	169.08

Table 4.3: Comparison of EVMS vs NMS algorithm

		EVMS			NMS		
$p_v = q$	$p_e = p$	T_m	pE	Time(s)	T_m	pE	Time(s)
DBLP							
0.05	0.01	20,469	-3.16	0.346	19,764	**-1.01**	0.32
0.1	0.01	30,944	-1.37	0.387	29,108	**0.66**	0.31
0.1	0.10	203,520	1.56	**0.544**	181,344	**-0.04**	0.76
Amazon							
0.01	0.01	10,415	-8.56	**0.37**	10,305	**-1.07**	0.41
0.15	0.10	190,579	0.74	**0.49**	170,482	**0.33**	0.74
0.05	0.15	183,062	2.03	**0.48**	171,088	**-0.49**	0.84
Berkeley-Stanford							
0.01	0.05	417,102	7.89	**10.35**	390,196	**-2.02**	27.03
0.05	0.1	1,147,570	-5.30	**31.99**	970,666	**-0.02**	92.26
0.1	0.1	1,592,198	-3.04	**47.61**	1,276,129	**-1.25**	186.67
Orkut							
0.01	0.01	2,283,760	-0.84	**38.10**	2,032,120	**-0.21**	94.97
0.01	0.05	7,725,722	1.71	**49.89**	6,999,753	**-0.33**	135.13
0.01	0.1	13,816,850	1.33	**113.89**	12,888,478	**0.20**	189.67

Table 4.4: Comparison of NMS vs Triest

		NMS			Triest	
$p_v = q$	$p_e = p$	T_m	pE	Time(s)	pE	Time(s)
DBLP						
0.05	0.01	19,764	**-1.01**	0.32	-9.20	1.87
0.1	0.01	29,108	**0.66**	0.31	-7.11	2.67
0.1	0.10	181,344	**-0.04**	0.76	-0.10	29.38
Amazon						
0.01	0.01	10,305	**-1.07**	0.41	54.61	0.8
0.15	0.10	170,482	**0.33**	0.74	-0.46	21.55
0.05	0.15	171,088	**-0.49**	0.84	-0.51	21.82
Berkeley-Stanford						
0.01	0.05	390,196	**-2.02**	27.03	2.33	751.08
0.05	0.1	970,666	**-0.02**	92.26	-3.20	3,203.47
0.1	0.1	1,276,129	**-1.25**	186.67	-4.87	4,560.59

Table 4.5: NMS vs Colorful triangle counting

	Colorful triangle counting			NMS		
p	T_c	pE	Time (s)	T_m	pE	Time (s)
			DBLP			
0.01	10,544	5.65	**0.52**	12,399	**-0.65**	0.71
0.08	80,930	5.35	0.61	80,631	**-0.52**	**0.45**
0.20	210,079	-0.90	1.08	200,610	**-0.49**	**0.67**
			As-Skitter			
0.05	556,308	7.04	5.71	547,546	**1.11**	**10.49**
0.10	1,110,313	-0.64	14.73	1,143,146	**-0.002**	**13.97**
0.20	2,218,433	-0.49	**44.30**	2,146,372	**0.53**	62.51
			Berkeley-Stanford			
0.05	332,018	**-0.30**	**8.68**	390,196	-2.02	27.03
0.10	664,925	-0.78	**22.27**	614,873	**-0.33**	53.72
0.17	1,109,374	0.48	**78.16**	1,185,173	**-0.0028**	155.80
			Orkut			
0.01	1,171,600	**-0.39**	**58.61**	1,575,696	0.92	72.87
0.03	2,929,852	-0.45	**56.47**	3,115,248	**-0.12**	122.29
0.10	11,717,612	0.08	**166.33**	13,386,721	**0.01**	350.92

CHAPTER 5. SUMMARY AND DISCUSSION

This thesis presents the Neighborhood Multi-Sampling algorithm, which is an extension to the Neighborhood sampling algorithm introduced before. It combines the ideas that sampling multiple edges and also sampling neighbors of the previous edges increases the probability to hit more triangles in the sample. The NMS algorithm computes (ϵ, δ)-approximation of the number of triangles in a graph by using expected space $O(mp)$ where p is $O(\sqrt{\frac{\Delta_E}{T}}\frac{1}{\epsilon}\sqrt{log\frac{\Delta_E}{\delta}})$.

Based on the experiments and theoretical results we can conclude that the performance of the NMS algorithm is way better, both theoretically and experimentally, compared to its counterparts (Neighborhood Sampling and EVSS). The NMS algorithm has better accuracy among all the algorithms studied here, though it is slower than the colorful triangle counting algorithm. The algorithm can also be adapted to continuously monitor the number of triangles and to work on dynamic graph stream. It would be interesting to extend these algorithms to other sub-graph structures other than the triangles.

BIBLIOGRAPHY

[1] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 1–16. ACM, 2002.

[2] Z. Bar-Yossef, R. Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 623–632. Society for Industrial and Applied Mathematics, 2002.

[3] L. Becchetti, P. Boldi, C. Castillo, and A. Gionis. Efficient semi-streaming algorithms for local triangle counting in massive graphs. In *KDD*, pages 16–24, 2008.

[4] L. Becchetti, P. Boldi, C. Castillo, and A. Gionis. Efficient semi-streaming algorithms for local triangle counting in massive graphs. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 16–24. ACM, 2008.

[5] E. Bott. *Family and Social Network: Roles, Norms and External Relationships in Ordinary Urban Families.* 1957.

[6] V. Braverman, R. Ostrovsky, and D. Vilenchik. How hard is counting triangles in the streaming model? In *ICALP*, pages 244–254, 2013.

[7] L. Bulteau, V. Froese, K. Kutzkov, and R. Pagh. Triangle counting in dynamic graph streams. *Algorithmica*, 76(1):259–278, 2016.

[8] L. S. Buriol, G. Frahling, S. Leonardi, A. Marchetti-Spaccamela, and C. Sohler. Counting triangles in data streams. In *SIGMOD*, pages 253–262, 2006.

[9] J.-P. Eckmann and E. Moses. Curvature of co-links uncovers hidden thematic layers in the world wide web. *PNAS*, 99(9):5825–5829, 2002.

[10] D. V. Foster, J. G. Foster, P. Grassberger, and M. Paczuski. Clustering drives assertively and community structure in ensembles of networks. *Phys. Rev. E*, 84:066117, 2011.

[11] M. M. Gaber, A. Zaslavsky, and S. Krishnaswamy. Mining data streams: a review. *ACM Sigmod Record*, 34(2):18–26, 2005.

[12] Z. Huang. Link prediction based on graph topology: The predictive value of generalized clustering coefficient. *Social Science Research Network*, 2010.

[13] M. Jha, C. Seshadhri, and A. Pinar. A space-efficient streaming algorithm for estimating transitivity and triangle counts using the birthday paradox. *TKDD*, 9(3):15:1–15:21, 2015.

[14] M. Jha, C. Seshadhri, and A. Pinar. Path sampling: A fast and provable method for estimating 4-vertex subgraph counts. In *WWW*, pages 495–505, 2015.

[15] H. Jowhari and M. Ghodsi. New streaming algorithms for counting triangles in graphs. In *COCOON*, pages 710–716, 2005.

[16] D. M. Kane, K. Mehlhorn, T. Sauerwald, and H. Sun. Counting arbitrary subgraphs in data streams. In *ICALP*, pages 598–609, 2012.

[17] N. Kavassery Parakkat. Triangle counting in graph streams: Power of multi-sampling. 2017.

[18] M. Latapy. Main-memory triangle computations for very large (sparse (power-law)) graphs. *Theor. Comput. Sci.*, 407(1-3):458–473, 2008.

[19] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.

[20] Y. Lim and U. Kang. MASCOT: memory-efficient and accurate sampling for counting local triangles in graph streams. In *KDD*, pages 685–694, 2015.

[21] M. Manjunath, K. Mehlhorn, K. Panagiotou, and H. Approximate counting of cycles in streams. In *ESA*, pages 677–688, 2011.

[22] K. Mousavi, N. K. Parakkat, and A. Pavan. Improved triangle counting in graph streams: Power of multi-sampling. *ASONAM*, 2018.

[23] S. Muthukrishnan. Data streams: Algorithms and applications (invited talk at soda'03), 2003.

[24] R. Pagh and C. E. Tsourakakis. Colorful triangle counting and a MapReduce implementation. *Inf. Process. Lett.*, 112(7):277–281, 2012.

[25] A. Pavan, K. Tangwongsan, S. Tirthapura, and K.-L. Wu. Counting and sampling triangles from a graph stream. *PVLDB*, 6(14):1870–1881, 2013.

[26] A. Sala, L. Cao, C. Wilson, R. Zablit, H. Zheng, and B. Y. Zhao. Measurement-calibrated graph models for social network experiments. In *WWW*, pages 861–870, 2010.

[27] T. Schank and D. Wagner. Finding, counting and listing all triangles in large graphs, an experimental study. In *Experimental and Efficient Algorithms*, pages 606–609, 2005.

[28] C. Seshadhri, A. Pinar, N. Durak, and T. Kolda. The importance of directed triangles with reciprocity: patterns and algorithms. *CoRR*, abs/1302.6220, 2013.

[29] L. Stefani, A. Epasto, M. Riondato, and E. Upfal. Trièst: Counting local and global triangles in fully-dynamic streams with fixed memory size. In *KDD*, pages 825–834, 2016.

[30] S. Suri and S. Vassilvitskii. Counting triangles and the curse of the last reducer. In *WWW*, pages 607–614, 2011.

[31] C. E. Tsourakakis. Fast counting of triangles in large real networks without counting: Algorithms and laws. In *ICDM*, pages 608–617, 2008.

[32] C. E. Tsourakakis, U. Kang, G. Miller, and C. Faloutsos. DOULION: counting triangles in massive graphs with a coin. In *KDD*, pages 837–846, 2009.

[33] C. E. Tsourakakis, P. Drineas, E. Michelakis, I. Koutis, and C. Faloutsos. Spectral counting of triangles via element-wise sparsification and triangle-based link recommendation. *Social Netw. Analys. Mining*, 1(2):75–81, 2011.

[34] C. E. Tsourakakis, M. Kolountzakis, and G. Miller. Triangle sparsifiers. *J. Graph Algorithms Appl.*, 15(6):703–726, 2011.

[35] K. Wolff, editor. *Selections from The Sociology of Georg Simmel*. 1950.