Approximating 4-Cliques in Streaming Graphs: The Power of Dual Sampling

by

Anmol Mann
B.Tech. (Computer Science), Chitkara University, 2018

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

Approximating 4-Cliques in Streaming Graphs:
The Power of Dual Sampling

by

Anmol Mann
B.Tech. (Computer Science), Chitkara University, 2018

Supervisory Committee

Dr. Alex Thomo, Supervisor
(Department of Computer Science)

Dr. Venkatesh Srinivasan, Co-Supervisor
(Department of Computer Science)

ABSTRACT

Clique counting is considered to be a challenging problem in graph mining. The reason is a combinatorial explosion; even moderate graphs with a few million edges could have clique counts in the order of many billions. When dealing with such big data, it becomes critical to not just analyze it, rather analyze it very efficiently. While randomized algorithms are known for estimating clique counts, 4-cliques have not received as much attention as triangles in the streaming setting.

In this work, we propose 4CDS, a fast and scalable algorithm for approximating 4-clique counts in a single-pass streaming model. By leveraging a combination of sampling approaches, we estimate the 4-clique count with high accuracy. We provide a theoretical analysis of the algorithm and prove that it improves upon the known space and accuracy bounds.

A comprehensive evaluation of 4CDS is conducted on a collection of real-world graphs. Our algorithm performs well on massive graphs containing several billions of 4-cliques and terminates within a reasonable amount of time. We experimentally show that our proposed method obtains significant speedup, outperforming several existing clique counting algorithms.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I would like to thank:

My family, for supporting me to pursue this extraordinary journey.

Dr. Thomo, for providing me this opportunity, and continuously motivating me to push my limits.

Dr. Srinivasan, for mentoring and sharing his knowledge.

Yudi Santoso for being a good friend, providing support throughout, and sharing his knowledge.

Eugen Toaxen for being a good employer, providing support throughout, and sharing his knowledge.

DEDICATION

This work is dedicated to my parents and brother!
Thanks for believing in me.

Chapter 1

Introduction

Clique listing and counts divulge important properties about the structure of massive graphs, especially social networks. Graphs are also popular in traditional enterprises for instance, a retailer might view the products and customers as a graph. Assume in a graph, we have a few nodes representing customers and the rest representing products. Edges are either formed between affiliated products or when customers purchase products.

Similarly, in financial institutions transactions can be modelled as a graph. In this case, we can assume that a fraction of nodes in our graph represent branches of the banks and the rest are the customers. The edges are formed when the customers deposit and withdraw money. Various types of queries can be performed on such graphs such as in the former example which product or classes of products are frequently bought together. Correspondingly, in the latter transaction graph, one might want to discover small deposits followed by large withdrawals to detect money laundering activities. Such queries can be expressed as finding all instances of a given subgraph of the actual graph. For instance, the first query can be considered as discovering all instances of a triangle constituting a customer and two correlated products. Similarly, the second query can look for a slightly more complex pattern like k-cliques or graphlets. Such queries are referred to as graph mining queries, where one is interested in locating structural patterns in the underlying graph.

In this research work, the problems related to mining patterns in massive graphs using approximation are studied. In addition to triangles, some of the commonly used patterns include *motifs*, which are subgraphs formed by common set of vertices and cliques. The standard approach to do this mining is to perform an iterative expansion. An iterative expansion is given a graph and a subgraph, we start with all instances

of the simplest subgraph that can be discovered (such as all the vertices in a graph). This is followed by iteratively expanding the candidate set by adding the neighbouring edges until the intended subgraph pattern is discovered. The challenge here is the amount of data which is generated. With more complex subgraph patterns such as 4-cliques, the intermediate data explosion results in this technique being intractable in massive graphs. Therefore, it becomes challenging to mine motifs such as 4-cliques, etc. in large graphs. On average, using iterative expansion it takes several hours to mine a subgraph pattern in a graph with 1 billion edges. However, using the technique proposed as part of this thesis work, 4CDS (4-Clique Dual Sampler), the same query can be processed on a much larger graph in a very short amount of time.

A clique in a graph is a set of nodes such that there is an edge between any two distinct nodes in the set. Specifically, a 4-clique is a set of four vertices, all connected to each other. Many recent works c.f. [37, 40, 46] make use of cliques in mining dense graphs to discover emerging dense sub-regions of a network. 4-cliques have also been shown to provide the foundation for computing the most practical case of nucleus decomposition of networks (see [38]). Notably, [38] provides in-depth analysis about the biological importance of small subgraph counts for detecting *network motifs*. Thus, clique listing and counts are considered to be very important in social network analysis and network science.

It is commonly thought that cliques, beyond three nodes, are difficult to enumerate or count. This is because the number of possible instances grows as $O(n^k)$, where k is the order of the clique and n is the order of the graph. Thus, for massive graphs, it is believed that algorithms that enumerate cliques or compute exact clique counts cannot terminate in a reasonable time [32], and thus, cannot scale well on large graphs. The clique counting problem has been studied extensively [3, 11, 23]. Alon et al. [3] proposed a technique to count given-length cycles. Bordino et al. [11] extends triangle counting to subgraphs on three and four vertices in a three-pass streaming model. Santoso et al. [36] present a method to exactly enumerate (not just count) four node graphlets (including 4-cliques) in static graph. Other proposed solutions, such as Arabesque [45] and 4-prof-dist [16] use distributed platforms.

We focus on the problem of estimating the count of 4-cliques using streaming algorithms. Specifically, we use the *one-pass streaming* model in which incoming edges of a graph are processed and the output updated as they come down a stream. In essence, once an edge moves down a stream, it cannot be processed again. Our algorithm processes edges of a data stream in a single pass. It is worth noting that

there have been plenty of studies on triangle count approximation in the streaming model. It was found that triangles can be estimated efficiently using state-of-art techniques such as [21, 40, 43].

When it comes to approximating the number of 4-cliques, to the best of our knowledge, none of the previous works can handle the one-pass streaming model. In this paper, we show that by leveraging a dual sampling of both edges and triangles we can achieve an accurate estimation of 4-cliques in the one-pass streaming model. Our proposed algorithm achieves a significantly improved run-time and is able to handle large graphs which other methods either take much longer to process or fail to complete processing at all. Moreover, unlike previous works, our solution is the first to work in a fully streaming setting producing the 4-clique count in only a *single* pass.

Our experiment shows that some massive graphs can be processed very fast, on the order of seconds, while some others take longer. We analyse the practical applicability of our method, and refine the feasibility limit of clique enumeration in general.

1.1 Graph pattern mining and Sampling

The motivation of many streaming models is *monitoring* a very large undirected graph that is changing dynamically. For example, a potentially massive graph can consist of communication between entities like IP addresses in a network. A graph is formed by a huge of volume of edges coming in as time progresses as for each connection there's an edge with IP addresses as it's vertices. Edges of the graph arrive in sequence. We want to continue to maintain a *unique* property of this large evolving graph. The context of this research revolves around counting sub-graphs, specifically, 4-cliques.

Pattern mining is the problem of finding instances of a given pattern such as triangles, 3-chain graphlets, k-cliques, network motifs, etc. in a graph. Most of the graph pattern mining algorithms can reveal interesting properties in a graph but struggle to process or scale massive graphs.

A baseline approach is to iterate over all possible embeddings in a graph. To do so, we begin with a vertex or an edge and subsequently, filter out those along the way that cannot possibly match. Then, one more vertex or an edge is added to expand the remaining candidate embeddings. This process continues to repeat itself to list all the patterns. The evident challenge in graph pattern mining, as opposed to graph analysis, is the exponentially large candidate set that needs to be checked.

1.2 Why 4-clique Counting?

Example : How often are four people connected via a single person in a network? This formation is referred to as a 4-Clique in graph theory. In other words, how friends of friends are connected to each other?

In general, there are many applications which makes use of clique counting. For instance, Becchett et al. [5]) talked about web spam detection. In [5], it's shown that a larger than usual number of triangles is an indicator of a web spam. A triangle is a subgraph pattern which constitutes of three vertices, all connected to each other.

If one wants to move to higher order cliques such as k-cliques, where k is the number of vertices, all of which are connected to each other in a graph, biological networks (Przul et al. [34], Kashtan et al. [24]) provides a graph summarization technique which is called as network motifs or graphlets. Network motifs are vectors consisting of several occurrences of various types of subgraphs in a large graph. These subgraphs are patterns such as edges, triangles, cliques of size 4, 5, etc. Therefore, network motifs are vectors providing a structural summary which describes a graph.

1.3 Motivation

Graphs are popular workflows in big data analytics. Most common applications of such datasets can be found in domains such as social networks, biology, and medicine. In particular, several small subgraph patterns in a network have been determined as the simple building blocks of complex biological networks. In fact, the distribution of their occurrences could reveal answers to many important biological questions [27,50]. 4-cliques, in particular, reveal community structure in protein interaction networks and word-association graphs [29].

There are many applications of *clique* discovery in the domain of Social Network Analysis (SNA). In social networks, a clique is described as a group of individuals who share similar interests and interact with one another. These groups are identified as *cliques* if every individual is directly tied to every other individual. Nodes in such a network tend to form highly connected neighborhoods which is measured by clustering coefficient. In other words, the degree to which nodes in a graph tend to cluster together is measured by clustering coefficient and is closely related to triangle estimation problem [31,39,44]. The next central subgraph pattern after triangles are *4-cliques*. To understand complex networks better, measuring clustering coefficients

of order 4 are crucial [51]. Evidence suggests that in most real-world social networks, nodes tend to create tightly knit groups characterized by a relatively high density of ties. Furthermore, the probability of such an event happening is greater than the the average probability of a tie randomly established between two nodes [19]. For example, when capturing the friendship relationship in Facebook graph, we obtain a higher value of clustering coefficient. The reason being cliques of friends are frequent in such types of social networks. To measure clustering coefficient of order 4, we need the count of 4-cliques in a network. This is the motivation behind this work.

Another motivation behind this work is based on the observations that in many mining tasks, one does not require the exact count of sub-graph instances. Often, in large graphs, it might be impossible to output all possible embeddings of a sub-graph pattern. Thus, in such scenarios it's favourable to leverage approximation to do pattern mining.

The main determination of our work is preserving *dual* subgraph patterns (edges and triangles) to obtain clique estimations. This reduces the execution times reasonably well and allows our algorithm to scale efficiently to large real-world graphs. Experiments on massive real-world graphs show that our streaming algorithm for counting 4-clique is practically fast, scalable and accurate.

1.4 Contribution

The main contributions of this thesis are:

- Using color coding (graph sparsification) and reservoir sampling, we present a one-pass, fast, and scalable streaming algorithm to approximate the number of *4-cliques* that significantly improves upon the state-of-art.
- We provide a detailed theoretical analysis and prove that our estimation is unbiased. Our algorithm works for any arbitrary edge ordering in a graph.
- Our algorithm does not need to wait for the entire graph stream to be processed to provide a 4-clique count. Rather, it can provide the 4-clique count of the graph seen so far at any instant of time.
- We create an efficient implementation of the algorithm for a single machine. For instance, our algorithm is able to run on the *densest* graph we consider,

dewiki, of millions of nodes and edges, within reasonable time and much more efficiently than other methods.

1.5 Organization

This thesis is organized as follows:

Chapter 1 provides a brief introduction about the relevance of graph mining and 4-clique counting, the motivation behind this research work and key contributions.

Chapter 2 provides the literature review for approximating k-cliques problem. It provides a brief overview of the major contributions made in the field of clique estimation. It also talks about the limitations and advantages of each approach.

Chapter 3 talks about the background knowledge for graph theory, streaming model, and goes into details of some sampling methods. It contains basic terminologies and concepts that are important to understand our contributions.

Chapter 4 presents the novel algorithm for approximating 4-clique count in a streaming one-pass model. Theoretical accuracy and complexity analysis of the algorithm is also provided in this chapter.

Chapter 5 contains the experimental results which evaluates the $4-clique$ approximation done on some massive real-world datasets. It also provides the comparison of our algorithm with the existing published methods.

Chapter 6 concludes the problem statement and the contributions made as part of this research work.

Chapter 7 contains a discussion about possible future work related to this area.

Chapter 2

Related Work

Extensive literature is available for clique counting in graphs using several computational models [13, 15, 27]. Our study focuses on the approximation algorithm using streaming model, however, not much has been done for counting k-cliques in that model.

Chiba and Nishizeki [13] is a seminal work on clique enumeration. They use degeneracy ordering to split graphs into several subgraphs and then recursively list cliques in these subgraphs. kClist is a parallel clique enumeration technique proposed in [15] which improves on [13] but still could not scale for large graphs. Moreover, the running time of kClist depends on the ordering of nodes. Primarily, the key idea behind discovering k-cliques is to count the number of $(k-1)$ cliques in the neighbourhood of every vertex in the graph. The limitation of kClist is that despite being to list trillions of cliques on a commodity hardware in a single *day*, it failed to scale up for larger graphs.

Milo et al. [27] analysed frequent subgraph patterns, and called them network motifs. Since then, there have been many studies on how to find and count small subgraphs within a graph or network, including those we already discussed in the Introduction. kClist is a parallel clique counting technique proposed in [15].

Most of the state-of-the-art techniques do not rely on streaming models to estimate k-cliques ($k >= 4$). Using streaming model as a technique to approximate clique counts in a graph can be tracked back to Bar-Yossef et al. [4] in 2002. To the best of our knowledge, there has not been a solution employing the streaming model to approximate 4-cliques by sampling both edges and triangles. Therefore, in this study we propose a novel streaming algorithm for counting 4-cliques.

For k-clique counting in graph networks, randomized techniques such as edge

sampling [35, 47, 49] and color coding [2, 6, 52] have been extensively used. The recently proposed ordering-based algorithm TURÁN-SHADOW [22] for estimating k-cliques ($k \leq 10$) (in a non-streaming setting) provides accurate clique estimates for large graphs.

Approximation algorithm proposed in [20] counts directed k-cliques ($k = 3, 4$) by decomposing graph network via node removal process. GUISE presented in [7] is based on Markov chain Monte Carlo uniform sampling strategy and random walk for estimating the motif frequency distribution of 3-node, 4node and 5-node motifs.

Note that *none* of the above works employs a streaming model. In other words, the aforementioned methods require multiple scanning of a graph to estimate a clique count. Thus, they cannot obtain a clique count at any instant in time. The limitation of these methods is that either they do not scale well on massive graphs or they cannot be employed on a single machine as they utilize clusters or parallelization techniques.

In contrast we consider a fully streaming setting and are able to produce an approximate clique count at any time instant. Our algorithm employs dual sampling (sampling both edges and triangles) based on the certainty that several sampling techniques have achieved fine results as demonstrated in [1, 14, 26, 28, 30]. Additionally, our algorithm does not take into account the ordering of nodes unlike [13, 15]. Our approach is based on the streaming model which not only results in optimizing resource utilization, but also allows computations in a single pass. Therefore, it allows our algorithm to scale well on large graphs with billions of edges.

Chapter 3

Preliminaries

In this chapter, we lay the groundwork and discuss several concepts, describe definitions and procedures related to k-clique counting. Sections 3.3, 3.4, 3.6 and 3.7 describe randomized clique counting algorithms closely related to our work in this study.

We consider simple and undirected graphs $G = (V, E)$ with no parallel edges and no self-loops. Graph G is a streaming graph in an *adjacency stream model*, where the edges are coming in a streaming fashion and their order is arbitrary. More specifically, G comes as a stream $\langle e_1, e_2, ..., e_{|E|} \rangle$ of edges. Let e_i denote the i-th edge in the stream order, $n = |V|$ denote the number of vertices, and $m = |E|$ denote the number of edges. The set of neighbors of a node $v \in V$ is $N(v)$ and the degree is $deg(v) = |N(v)|$.

A 4-clique is a subgraph of four nodes where each node is neighbor of every other node. It has 6 edges and four triangles as shown in Figure 3.1. The notations used in this study are described in Table 3.1.

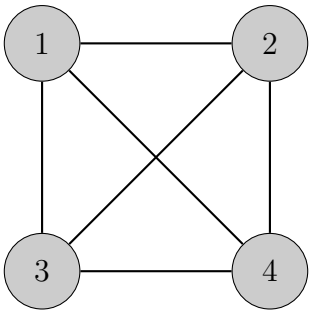


Figure 3.1: A 4-clique subgraph

Table 3.1: Notation used in this paper.

Symbol	Definition
$G(V, E)$	An undirected graph with set of nodes V and set of edges E.
DAG	An acyclic directed graph.
$n = \|V\|$	Number of vertices in G.
$m = \|E\|$	Number of edges in G.
Σ	A stream of edges $\langle e_1, e_2, ..., e_{\|E\|}\rangle$, where e_i denotes the i^{th} edge in the stream.
$\Sigma(G)$	A stream of edges of graph G.
$e = \{u, v\}$	An undirected edge between u and v
$N(u)$	Set of neighbours of vertex u.
$N(u)^+$	Set of out-going neighborhood of vertex u in a DAG
$d(u)$	Degree of vertex u, $d(u) = \|N(u)\|$.
$\mathcal{N}(e)$	Set of edges adjacent to edge e and it arrives after e in the stream.
c	Palette of colors to choose from
c_v	A random number in $[1, c]$
$(u, v, w)_\triangle$	A triangle with nodes u, v, and w
$(u, v, w)_\angle$	A wedge with nodes u, v, and w, centered at v
C	Exact number of 4-cliques.
T	Estimate number of 4-cliques.

3.1 Data Streaming Model

A data stream constitutes a sequence of data items in which the items arrive one at a time. The data streams studied in this work are graph streams. The adjacency stream model is a streaming graph model in which a given graph $G(V, E)$ is rendered as a stream of edges $\Sigma = \langle e_1, e_2, ..., e_{|E|}\rangle$. Each edge e_i arrives in an arbitrary order at any instant in time.

Lots of resources are wasted to process an entire graph stream as the data items can be accessed multiple times in case of multiple *passes*. Due to limited memory availability and resources, exact or an 100% accurate count is not the best way forward. Streaming approach provides clique counts instantly on the arrival of an edge. This reduces the consumption of resources. Hence, approximation counting (with error bounds) provide an optimal solution to our problem. The objective is to trade-off accuracy for faster results. Approximate analytics is an area that has gathered attention in big data analytics. In fact, many social network firms use a count of similar

Algorithm 1 BRUTEFORCECLIQUECOUNT(G, k)

1: **if** $k == 1$ **then**
2: **return** $|V|$
3: **if** G is a 4-clique **then**
4: **return** $\binom{|V_G|}{k}$
5: Let $cliqueCount = 0$
6: Convert G into a DAG directed graph by ordering the vertices according to degeneracy ordering.
7: Let $N(v)^+$ denote the out-neighborhood of vertex v in DAG
8: **for** each vertex $v \in V$ **do**
9: $cliqueCount = cliqueCount + BruteForceCliqueCount(N(v)^+, k-1)$
10: **return** $cliqueCount$

graphlets in their applications to analyse social graph similarity. The context of this research work revolves around employing a randomized approach to count 4-cliques based on the streaming model.

3.2 Brute force clique counting technique

Chiba and Nischizeki provided a brute force clique enumeration technique that uses degeneracy ordering to divide the graph into various subgraphs. The cliques are counted recursively in these subgraphs. The algorithm lists every k-clique in a graph and a 4-clique counting version of the algorithm is described in Algorithm 1.

3.3 Color Coding

We use the color-coding technique 2 introduced by Alon et al [2]. This is a randomized approximation algorithm which cuts back the search space for cliques and speeds up the clique counting process. Each vertex v of G is assigned a color c_v which is a random number in $[1, c]$. This technique sparsifies the graph G by assigning random color to each vertex and preserves an edge if the colors of its two endpoints are the same.

In Algorithm 2, after assigning colors to all vertices, we then count the number of multicolored k-cliques ($c = k$, in this approach). Multicolored implies that each vertex of the clique has a different color. Every vertex of a clique receives a different color with a probability $c!/c^c$. Hence, the expected number of multicolored 4-cliques

in a graph would be $C \times c!/c^c$. We can count the multicolored cliques and compare it with the expected value to obtain an estimate for the number of 4-cliques C'. The limitation of color-coding technique is that accuracy gets traded off at a large scale when the number of multicolored cliques decreases.

Algorithm 2 ColorCoding(G, c)

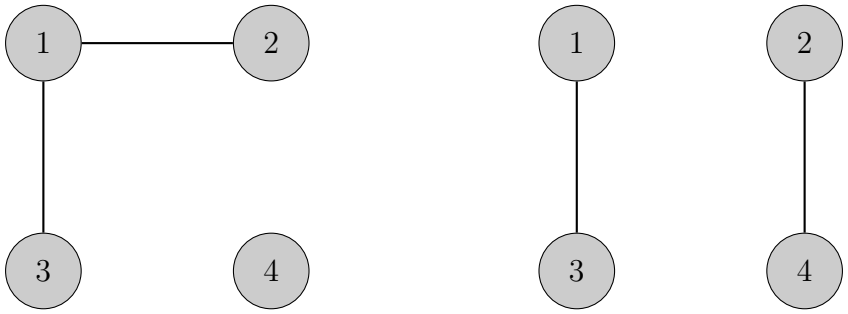
1: Assign a color to all vertices in V ▷ c_v is a random number in $[1, c]$
2: Let $color[v] = c_v$ denote the color of vertex v
3: $C' =$ perform recursive color coding on subgraphs of the sparsified graph G
4: $cliqueCount = C' \times c!/c^c$
5: **return** $cliqueCount$

3.4 Reservoir Sampling

Reservoir sampling [49] is a collection of randomized sampling algorithms used for randomly choosing a sample of x items from a list G' containing s items, where s is either an unknown number or is a very large number that does not fit in memory. It selects a data item to be processed or not based on a probability. This is a widely accepted sampling technique in the context of triangle counting in graph networks. Although, this approach is flexible enough and can be employed for large patterns as well as shown in [16]. The key insight is that each incoming data item in the stream has the same probability to be preserved as the others. The algorithm shown in method 3 is employed to explicitly count 4-cliques. Each edge is sampled with some probability r. Then, the 4-cliques are enumerated in the induced subgraph formed by the sampled edges. A 4-clique is detected only if all it's 6 edges survive in the sampled subgraph. The probability of such an event happening is $r^{\binom{4}{2}}$. The count is then scaled up by $1/r^{\binom{4}{2}}$ to obtain an unbiased estimate for number of 4-cliques. Much like color-coding technique described in Algorithm 2, this technique also poorly in terms of accuracy when $k > 3$.

Algorithm 3 EdgeSampling(G, r)

1: **for each** edge $e_i \in E$ **do**
2: remove e_i from G with sampling ratio r
3: **return** BruteForceCliqueCount$(G, 4)/r^{\binom{4}{2}}$



(a) Type I (b) Type II

Figure 3.2: Type of 4-cliques estimated by Neighborhood Sampling

3.5 Neighborhood Sampling for 4-Cliques

One of the most recent *extensively* researched sampling techniques is Neighbourhood sampling [30]. This approach was proposed in the context of a specific graph pattern, triangle counting. It also provides an approach for estimating 4-cliques, however, there is no experimental analysis provided in [30]. The central objective of neighborhood sampling is to sample one edge from an edge stream, and then moderately add more edges until either they form a 4-clique, or it becomes impossible to form the pattern. Prior to counting 4-cliques, the set is partitioned into two classes (Type I and Type II) as shown in Figure 3.2. Two separate neighborhood sampling methods are provided to estimate these two classes of 4-cliques. Type I refers to a 4-clique in which the first two incoming edges of a 4-clique share a common vertex. Otherwise, the 4-clique represents a Type II clique.

For estimating 4-cliques of Type I, the algorithm works as follows. Given a graph with $|E|$ edges, a sample of three edges over four vertices is maintained by the sampling algorithm. Then, the algorithm looks for the edges to complete the 4-clique formation. To sample these three edges, the algorithm uses standard reservoir sampling.

To estimate 4-cliques of Type II, we only need to randomly sample two edges. Then, the algorithm waits for the remaining edges to complete the 4-clique with the two sampled edges. The drawback of employing neighborhood sampling for estimating 4-cliques is that it is really slow. The reason being the algorithm tries to look for all common neighbors for the edges sampled at first level.

3.6 Graft

GRAFT proposed in [35] is a variant of edge-sampling algorithm. It is an approximate counting algorithm, which counts the occurences of all graphlets having up to five vertices.

The sampling method proposed by Rahman et al. samples edges uniformly and randomly from a graph G. Subsequently, the algorithm iterates over each of the sampled edges in a sequential manner. Then, the number of graphlets (subgraph patterns) that each sampled edge contributes to is calculated (line 5). Note that this number is the partial count, which is incremented sequentially as the sampled edges are processed. The partial count is normalized (line 6) to obtain the total count of the clique in G to avoid counting the same clique more than once. The count is then scaled up by dividing it with the sampling ratio (line 7). Algorithm 4 presents a 4-clique counting version of the algorithm.

Algorithm 4 GRAFT(G, r)

1: Let $cliqueCount = 0$, g denote a 4-clique, e_g be a specific edge in g, n_f be the normalization factor for g
2: Sample edges E_r from E with probability r ▷ without replacement
3: **for each** $e_i = (u, v) \in E_r$ **do**
4: align e_g with e
5: l_i = Enumerate all listings of g found in G ▷ where e_i and e_g are aligned
6: $cliqueCount = cliqueCount + (l_i/n_f)$
7: **return** $cliqueCount/r$

3.7 Turán-Shadow

TURÁN-SHADOW [22] is an approximate k-clique ($k <= 10$) counting algorithm based on the classic Turán theorem [48]. The graph G is first decomposed into smaller dense subgraphs (shadows). Then, cliques within these dense subgraphs are sampled to provide an estimate. A key concept in this algorithm is that of *clique shadows*. Intuitively, a shadow is a large collection of subgraphs, in which the sum of clique counts corresponds to the total clique count of G. We drill down on these shadows and count cliques within them.

- **Degeneracy**

Algorithm 5 ShadowConstruction(G,k)

$\tau \leftarrow \{(V, k)\}$, $S \leftarrow \phi$
while $\exists (H, l) \in \tau$ s.t. $\rho(H) \leq 1 - 1/(l-1)$ **do**
    Let G_H be the subgraph of G induced by H
    $\vec{G_H} \leftarrow$ construct a DAG by the degeneracy ordering on G_H
    Let $N(v)^+(\vec{G_H})$ be the set of out-going neighbors of vertex v in $\vec{G_H}$
    for each $u \in H$ **do**
        if $l \leq 2$ or $\rho(N(v)^+(\vec{G_H})) > 1 - 1/(l-2)$ **then**
            $S \leftarrow S \cup \{(N(v)^+(\vec{G_H}), l-1)\}$
        else
            $\tau \leftarrow \tau \cup \{(N(v)^+(\vec{G_H}), l-1)\}$
    $\tau \leftarrow \tau \setminus \{(H, l)\}$

Definition 3.7.1. The degeneracy of a graph is a measure of the density of the graph. It is the smallest value d such that any induced subgraph of the graph has a vertex with degree at most d.

- **Degeneracy ordering**

  Definition 3.7.2. The degeneracy ordering of a graph is an ordering of vertices obtained by taking the lowest degree vertex in the graph at that point (ties may be broken by id) and removing it from the graph.

  Given a graph G with degeneracy d, a node ordering is called a degeneracy ordering if every node in G has $\leq d$ neighbors which come later in the ordering. This technique is utilized by Turán-Shadow to convert a graph into a DAG.

- **Arboricity**

  Definition 3.7.3. Arboricity, denoted by α, is used to measure the sparsity of a graph G.

  The minimum number of forests into which the edges of graph G can be partitioned is defined as arboricity of G.

Algorithm 7 involves two sub-procedures: shadow construction 5 and sampling 6. In the first procedure, the construction of a clique shadow takes place. We convert a graph G to a DAG directed graph, order by degeneracy. Then, we build a clique enumeration tree, stopping whenever Turán density (the density bound for k) is reached.

Algorithm 6 SAMPLING(S, t)

1: $X \leftarrow \phi$ ▷ indicator set
2: $w(H) \leftarrow \binom{|H|}{l}$ for each $(H, l) \in S$
3: $p(H) \leftarrow w_H / \sum_{(H,l)\in S} w(H)$
4: **for** $j = 1$ to t **do**
5: Independently sample (H, l) from S based on probability $p(H)$
6: $R \leftarrow$ randomly picking l vertices from H
7: **if** R forms a l-clique **then**
8: $X_r \leftarrow 1$
9: **else**
10: $X_r \leftarrow 0$
11: **return** $\frac{\sum_r X_r}{t} \times \sum_{(H,l)\in S} w(H)$

This provides us with a k-clique Turán shadow of G. Specifically, the Turán theorem states that for any graph G, if the density of G, denoted by $\rho(G) = |E|/\binom{|V|}{2}$, is greater than $1 - 1/(k-1)$, then G contains a k-clique. For a given k, the Turán shadow contains a set of (H, l) pairs, where $H \subseteq V$ is a subset of vertices and $l \leq k$ is an integer. Line 2 of the shadow construction shows that for each $(H, l) \in S$, the density of the subgraph induced by H $(\rho(G_H))$ is larger than Turán threshold $1 - 1/(l-1)$. Hence, for a (H, l) pair, the subgraph G_H contains a l-clique by Turán theorem.

Shadow construction algorithm 5 works as follows: pick a (H, l) pair iteratively that does not satisfy Turán threshold (line 2). Then, based on the degeneracy ordering, construct a DAG directed graph with (H, l) (lines 3-4). For any pair $(N(v)^+(\vec{G_H}), l-1)$ meeting the threshold is added to the Turán shadow S else it is added to τ (lines 7-10). At last, the algorithm removes (H, l) from τ (line 11) before re-cursing back to line 2. The intuition behind this algorithm is to iteratively refine the pairs in τ until all pairs satisfies the Turán threshold.

Algorithm 7 TURÁN-SHADOW(G, k, ϵ, δ)

1: Compute clique shadow $S =$ SHADOWCONSTRUCTION(G, k)
2: Set $\gamma = 1/max_{(S,l)\in S}(f(l)|S|^2)$
3: $t = (20 \cdot \epsilon^2/\gamma) \cdot log(1/\delta)$ ▷ number of samples required for estimation
4: **return** SAMPLING(S, t)

Once we have the Turán shadow, we set up distribution over sets in Turán shadow as detailed in Algorithm 6. We pick a set from this distribution, pick some subset

of random vertices, and check if they form a clique. We then scale the success ratio (probability of finding a clique). In Turán-Shadow, bulk of the time is spent in building the tree, and only a small fraction is needed for sampling. The drawback of Turán-Shadow is that it requires an entire shadow to be available for sampling, which requires considerable space.

Chapter 4

Algorithms and Analysis

This chapter presents a randomized algorithm called 4CDS (4-Clique Dual Sampler) to approximate the 4-clique count in a streaming one-pass model. Essentially, it employs dual sampling (sampling both edges and triangles) based on the fact that such sampling techniques have been used with much success as demonstrated in [1, 14, 26, 28, 30]. We provide the pseudo-code and then go into the details of the algorithm analysis to show runtime and memory bounds.

4.1 Problem Statement

In this work, we address the problem of estimating the counts of global *4-cliques*. We assume the standard data stream model where the insertions 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. Given an undirected graph $G = (V, E)$, a 4-clique is a set C of four vertices in V with all pairs in C connected by an edge. The problem is to count the number of 4-cliques in G in a one-pass streaming model.

At this point, we have several questions to answer:

RQ1. How to sample both edges and triangles from a graph stream? Which sampling techniques do we need to employ?

RQ2. How to count 4-cliques in a sampled subgraph?

For counting 4-cliques in sampled subgraphs induced by the sampled triangles, we use a standard counting procedure 9 (as described in section 4.3).

RQ3. How to estimate the true 4-clique count?

Our proposed solution should provide unbiased estimates. In other words, the difference between the true 4-clique count and the expected value of its estimate should be really small, thus, approaching zero bias.

In this research work, we focus on answering RQ1 and RQ3.

4.2 Main Contributions

Our main theoretical result is a randomized one-pass streaming algorithm 4CDS that approximates the 4-clique count. The algorithm is implemented on a single commodity machine and obtains clique counts for various datasets, the largest of which has 50M edges. The main contributions of our work are as follow:

1. Using color coding (graph sparsification) and reservoir sampling, we present a one-pass, fast, and scalable streaming algorithm to approximate the number of *4-cliques* that significantly improves upon the state-of-art. The key idea behind this approach is to utilize the benefit of sampling both edges and triangles in a graph stream.

2. We provide a detailed analysis (in Section 4.4) and prove that our estimation is unbiased (in Theorem 1). To obtain unbiased estimates, the discovery probability needs to be computed exactly. Instead of using only the edges stored in memory, we use every arrived edge to improve its estimation, even if the edge is about to be discarded without being stored.

3. Our algorithm works for any type of edge ordering in a graph. Hence, there is no need of any pre-processing of graph streams.

4. Our algorithm does not need to wait for the entire graph stream to be processed to provide a 4-clique count. Rather, it can provide the 4-clique count of the graph seen so far at any instant of time.

5. We create an efficient implementation of the algorithm for a single machine. For instance, our algorithm is able to run on the *densest* graph we consider, dewiki, of millions of nodes and edges, within reasonable time and much more efficiently than other methods.

Algorithm 8 4CDS (4-Clique Dual Sampler)

Input: Graph stream G

```
1: global variables: cliqueCount ← 0, T ← ∅
2: local variables: S ← ∅, number of colors c
3: for each edge e = (u, v) ∈ G do
4:     Count4clique(u, v)
5:     colorU ← random_int(0, c)
6:     colorV ← random_int(0, c)
7:     if colorU == colorV then
8:         Insert edge e in S
9:         uSet ← N(u, S)                          ▷ neighbors of u in S
10:        vSet ← N(v, S)                          ▷ neighbors of v in S
11:        for each t ∈ uSet ∩ vSet do
12:            if coin_toss(r) == "heads" then
13:                Insert triangle (u, v, t) in set T
```

4.3 Algorithms

Our algorithm for 4-clique approximation, denoted by 4CDS (4-Clique Dual Sampler), is a sampling algorithm (Algorithm 8) that uses color coding and reservoir sampling. 4CDS first updates its estimate using the incoming edge and previously sampled edges. For each observed 4-clique addition, increase corresponding estimate by the reciprocal of the probability that the 4-clique is discovered. After that, 4CDS decides whether to sample that edge and the corresponding triangles formed by that edge or not.

The algorithm works as follows: Initially, the algorithm sets global variables, $cliqueCount = 0$ and set $T = \emptyset$, and local variables, set $S = \emptyset$ and c (lines 1-2). The preserved edges are stored in set S, and the sampled triangles in set T. Then, the algorithm iteratively processes every incoming edge e in the graph stream G. Firstly, 4CDS invokes sub-procedure Count4Clique (Algorithm 9) to count the number of 4-cliques formed by e with the induced graph by the sampled triangles in T. By iterating over the common neighbors of vertices u and v, denoted by w, we discover 4-cliques formed by edge e by finding the common neighborhood between u, v and w (line 6). The count is then scaled up as shown in line 9 of Algorithm 9. Specifically, to obtain an unbiased estimate, we scale our count by r^2/c^3, which is the probability of discovering a 4-clique (will be explained further in Section 4.4 below).

After counting the 4-cliques formed by edge e, we uniformly and randomly assign

Algorithm 9 COUNT4CLIQUE

Input: Edge $e = (u, v)$

1: $count \leftarrow 0$, $scale \leftarrow c^3/r^2$
2: $uSet \leftarrow N(u, T)$, $vSet \leftarrow N(v, T)$
3: ▷ neighbors of vertices u and v in T
4: **for** each common neighbor $w \in N(u, T) \cap N(v, T)$ **do**
5: $wSet \leftarrow N(w, T)$ ▷ neighbors of w in T
6: $cSet \leftarrow wSet \cap uSet \cap vSet$
7: $count = |cSet|/2$
8: **if** $count > 0$ **then**
9: $cliqueCount \leftarrow cliqueCount + (count * scale)$

colors to both endpoints of e (lines 5-6). The algorithm computes the color of each vertex upon the arrival of each edge e, this takes $O(m)$ work in total, and $O(m)$ space. Then, 4CDS determines if e is preserved or not (line 7) by matching the colors of vertices u and v. If they are a match, edge e is preserved, that is, it is added to the set of sampled edges S (line 8); else the algorithm continues with the next incoming edge (line 3). Subsequently, after preserving e, the algorithm iterates through all triangles formed by e on the subgraph induced by S (line 11). These triangles are then sampled in T uniformly at random using reservoir sampling with probability r (lines 12, 13). Recall T is the set of sampled triangles stored in the reservoir.

We note that 4-cliques are discovered (i.e., counted) in line 4 upon arrival of each edge e before e is either sampled or discarded. The main idea behind this is to reduce the estimation error and make the algorithm more robust by minimizing the loss of information. In other words, we utilize every edge regardless of whether it will be preserved or discarded. This significantly increases the discovery probability of a 4-clique and reduces the variance of the estimation.

4.4 Analysis

Theorem 1. 4CDS (Algorithm 8) provides an unbiased estimate of the number of 4-cliques in an undirected graph stream at any time instant.

Proof. Let C be the true 4-clique count in graph stream G, F be the 4-clique count in the sub-graph T (induced on sampled triangles), $p = r^2/c^3$ (sampling probability to discover a 4-clique), and $X = F/p = F \cdot (c^3/r^2)$. Note that this is the estimate provided by our algorithm, 4CDS, for the number of 4-cliques.

We now show that $E[X] = C$. Let $S = \{s_1, \ldots, s_C\}$ be set of 4-cliques present in the stream in that order. Let F_i be an indicator variable denoting whether s_i is discovered or not. Clique s_i is discovered if (1) all the four vertices are assigned the same colour and (2) The first two triangles of s_i in the stream are sampled (the other two triangles are discovered when the sixth edge of s_i arrives). We now compute the probabilities of (1) and (2).

Each vertex is assigned a color c chosen uniformly and randomly (lines 5-6), and the edges are preserved if both endpoints have the same color. For a 4-clique to be discovered, all four vertices must be assigned the same color. The probability of such an event happening is $1/c^3$. This is because we first need to fix the color of vertex u (we assign color c to vertex u). Then, we pick the colors for remaining three vertices and the probability to assign a color c to each vertex is $1/c$. Hence, the probability of all three remaining vertices are assigned the same color as that of vertex u is $1/c^3$.

All four triangles in the 4-clique sub-structure need to be discovered for a 4-clique to be found. Once the first two triangles of a clique are sampled, the arrival of the final edge e_6 will form two more triangles and complete the formation of a 4-clique as shown in figure 4.1.

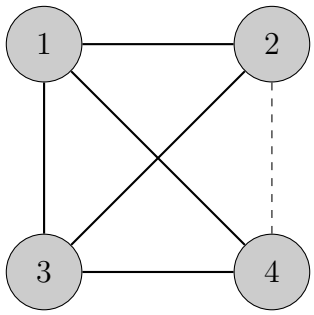


Figure 4.1: A 4-node graphlet waiting for the arrival of final edge $\{2, 4\}$ to complete a 4-clique

With respect to triangles, we sample the first two triangles of of clique using reservoir sampling. Each triangle is sampled uniformly and randomly with a sampling ratio r. Hence, the probability to sample the first two triangles is r^2. Note that these two events, color coding of vertices and sampling of triangles, are independent of each other.

Now, the number of 4-cliques in graph T is $F = \sum_{i=1}^{C} F_i$. Therefore, $E[F] = E[\sum_{i=1}^{C} F_i] = \sum_{i=1}^{C} E[F_i] = \sum_{i=1}^{C} r^2/c^3 = C \cdot r^2/c^3$. Finally, $E[X] = E[F \cdot (c^3/r^2)] = (c^3/r^2) \cdot E[F] = (c^3/r^2) \cdot C \cdot r^2/c^3 = C$. Thus, $E[X] = C$. ■

Theorem 2. 4-Clique Dual Sampler 8 takes $O(m/c + K)$ storage space, where K is the reservoir size for storing sampled triangles.

Proof. Color-coding maintains $O(m/c)$ edges on average, while reservoir sampling maintains up to $O(K)$, where K is the fixed memory budget. Therefore, the average storage cost of our algorithm becomes $O(m/c + K)$ at the end of a graph stream. This makes 4CDS a space-efficient one-pass streaming algorithm. ■

Theorem 3. Algorithm 8 takes $O(d + Kd + d^2/c)$ expected times to process m edges in the graph stream G.

Proof. Let $N[i]$ be the neighbors of vertex i, s is the number of sampled edges in S and t be the number of sampled triangles in T. Each edge is processed by the algorithm before it can be discarded. In other words, each edge contributes to the 4-clique count. Hence, the algorithm 8 first iterates over each sampled triangle to discover all 4-clique formed by incoming edge e.

The most expensive step in processing each edge is to intersect neighbors of u and v in T (line 4 algorithm 8) and neighbors of u, v and w (line 6 in algorithm 9). This takes $O(1 + N(u) + N(v) + N(w)) = O(1 + E[|T|]) = O(1 + K)$, where K is the fixed reservoir size for sampling triangles in-memory. Let us assume that the algorithm has processed d data items so far. Thus, processing first d elements takes $O(d + K \cdot d)$ on average.

Further, we try to sample e using color coding sampling technique. If e is sampled, the algorithm iterates over all the edges stored in S looking for a triangle. This step takes $O(E[N(u)+N(v)]) = O(E[|S|]) = O(s/c)$. Thus, in total, the triangle sampling step takes $\sum_{s=1}^{d}(s/c)$ time. This turns out to be $O(d^2/c)$ on average. Therefore, to provide an estimate of the global 4-clique count, our algorithm 8 on average takes $O(d + Kd + d^2/c)$ time overall. ■

4.4.1 Time Complexity Comparison

We provide a detailed time analysis of how and why 4CDS algorithm is faster than its competitors. The time complexities of Graft [35] and Turán-Shadow [22] are

discussed below.

Time Complexity for Graft: In this algorithm, firstly, the edges are sampled in memory without replacement. Thus, implying no fixed memory budget is employed by GRAFT. The fixed sampling rate r specified as an input parameter implies that the number of edges sampled by the GRAFT algorithm will be $p \cdot |E|$. The algorithm then iterates through each of these sampled edges to count the number of occurrences of each clique that uses that sampled edge. This turns out to be the most expensive step of the algorithm. The reason being there is no separate method to compute 3−, 4− or 5− cliques separately. In addition, as there is no memory budget, the sampled edges keeps on increasing depending upon the number of edges in the graph. Therefore, all these factors combined make GRAFT infeasible to process large graph. Our experimental analysis in 5 show that GRAFT is not scalable and cannot be employed on massive datasets.

Assuming d to be the highest degree of a node, the number of possible ways to get 4-clique embeddings from an edge are at most $3 \cdot d$. .Hence, the time complexity of the algorithm is dominated by the nodes with highest degrees. Thus, the expected cost of work for $Graft$ to estimate the clique count becomes $O(2 \cdot d * 3 \cdot d * (\binom{4}{2} - 3) * |E|) = O(2 \cdot d * 3 \cdot d * 3 * |E|) = O(d^2 \cdot |E|)$. Note that after listing an embedding, existence of $\binom{4}{2} - 3 = 3$ edges are checked to discover the type of clique embedded.

Time Complexity for Turán-Shadow: The running time of the algorithm depends on the size and time to construct the Turán shadow (explained in Section 3.7). Hence, only after the shadow is constructed, the sampling of cliques takes place. This makes TURÁN-SHADOW a non-streaming algorithm as for constructing the shadow first, the whole graph needs to be available before sampling can take place. The theoretical time bound of TURÁN-SHADOW is $O(n \cdot \alpha^3)$, where α is the arboricity of graph G.

Comparison: On comparing the runtime complexities of 4CDS, GRAFT and TURÁN-SHADOW, we can infer that the worst-case time complexity of GRAFT is typically higher than that of TURÁN-SHADOW and 4CDS. Experimental results in Chapter 5 present the validity to our inference. TURÁN-SHADOW being an ordering-based algorithm takes time to compute the ordering and shadow before it can sampled cliques. Compared to TURÁN-SHADOW, 4CDS algorithm has a better time complexity as it is not dependent on the ordering on nodes and does not require any pre-processing time. However, once TURÁN-SHADOW constructs the shadow, the sampling is done in linear time.

4.4.2 Space Complexity Comparison

A comprehensive space analysis of 4CDS, Graft [35] and Turán-Shadow [22] is provided as follow. Note that our algorithm does not store entire graph in-memory and hence, stores only a fraction of edges and triangles in the reservoir.

Space Complexity for Graft: The authors of [35] did not provide an analysis for space usage by the algorithm as it is very hard to compute. Our analysis of Graft shows that on average it takes $O(|E| + |V|^2)$ the expected amount of space. The reason behind this is that the algorithm maintains two separate data structures, one is the adjacency list representation of graph G and the other is the hash-table to check if an edge exists between two vertices. This makes Graft infeasible to scale on large graphs as it always has to store entire graph G in memory. The reason behind storing G in memory is to count the generation tree graphlet. In [35], it is shown that to process a graph with 1.7M vertices and 11M edges, the algorithm took around 4 GB of RAM.

Space Complexity for Turán-Shadow: To maintain the Turán shadow structure, the algorithm proves to be really memory intensive as it takes $O(m + n \cdot \alpha^2)$ amount of expected space on average. This is because the shadow is relatively larger than the graph size. Our experimental analysis shows that, for a graph with 770K vertices and 6M edges, the algorithm takes around 5GB of memory space. Hence, it is clear that the algorithm cannot scale well for massive datasets due to considerable overhead.

Comparison: After comparing the space complexities of Turán-Shadow, 4CDS and Graft, we can claim that Turán-Shadow performs the worst as it takes large amount of space to maintain the Turán shadow structure. It is followed by Graft, which always stores entire graph in memory to enumerate tree graphlets. To conclude, our algorithm 4CDS, takes less space than both Graft and Turán-Shadow.

4.4.3 Parameter Selection

Prior to selecting input parameters for 4CDS, we need to address the following questions.

a. Will 4CDS ever run out of memory like its competitor Graft [35]?

4CDS employs reservoir sampling with replacement. In other words, once the reservoir is full, a new triangle is sampled by replacing an existing sampled

triangle. Thus, our method limits the size of sampled triangles in-memory by fixing the memory budget = K, which is similar to algorithms using reservoir sampling technique with replacement. On the contrary, in algorithms such as GRAFT, there is no such storage limit and the reservoir size for storing preserved edges always grows. Hence, we overpower the biggest limitation of state-of-the-art algorithms by having an assigned budget.

b. Does 4CDS scale well on massive datasets?

Algorithms such as GRAFT [35], TURÁN-SHADOW [22], etc., which may run out of memory since at some point (as detailed in 4.4.2). Our experimental results in section 5.4.2 also proves our claim. Therefore, this implies that these state-of-the-art methods have limited scalability. As 4CDS has a fixed memory budget for sampling triangles, which implies that it does not run out of memory as discussed in **a.**. Hence, our algorithm do scale well on large graphs.

c. Does 4CDS suffers from information loss?

Algorithms without a memory budget tend to discard edges even when the number of edges preserved in-memory are less than an ideal threshold. Due to this issue, some algorithms do suffer from loss of information. 4CDS alleviates this problem as it tries to maintain as many triangles as possible in the reservoir at all times.

Table 4.1 details the input parameters required for our algorithm and existing state-of-the-art algorithms, GRAFT and TURÁN-SHADOW. 4CDS has two input parameters, the number of colors c and the sampling probability r. There are a few limitations of 4CDS algorithm in terms of determining the value of input parameters beforehand. These limitation are discussed as follows.

- The values of input parameters, c and r, in advance are hard to configure and require thorough analysis in order to get the desired estimation quality. It also depends on the user, i.e., how much accuracy is the user willing to trade-off to achieve faster execution times.
- As we use reservoir sampling with replacement, therefore, if both the chosen value of r and the memory budget are large enough, 4CDS may run out of memory for large datasets.

	GRAFT	TURÁN-SHADOW	4CDS
Input Parameters	Sampling rate r	Number of samples K Saturation rate γ	Number of colors c Sampling rate r Memory budget K
Probability to sample edge	r	N/A	$1/c$
Probability to sample triangle	N/A	N/A	r
Probability to sample cliques in a shadow	N/A	$O(1/\gamma)$	N/A
Use Reservoir Sampling?	✗	✗	✓
Use Random Sampling?	✓	✓	✗
Have memory growth limits?	✗	✗	✓

Table 4.1: Characteristics of GRAFT, TURÁN-SHADOW and 4CDS algorithms

- On the contrary, if the chosen value of r small, the algorithm will provide skewed estimates.

Further, as our algorithm requires sampling rate r (for preserving triangles) as an input parameter, the user needs to decide the number of triangles that are to be stored in-memory. Usually, in real-world scenarios, the graph properties are unknown beforehand, the user might provide an arbitrary value and if the value is too small with respect to the true value of the triangle count, the estimate will be inaccurate.

For real-world graphs following power-law degree distribution (especially social-networks), it is hard to predict in advance what values of these parameters will yield what kind of accuracy. This is because the partial count distribution of cliques in these graphs are irregular.

Chapter 5

Experimental results

In this chapter, we discuss evaluation experiments and results for the algorithm 4CDS proposed in the previous chapter. The experimental study has been performed on undirected graphs and the comparison is provided between our algorithm and state-of-the-art algorithms (discussed in Section 3). Section 5.1 provides the details of various datasets used for experimental analysis. In Section 5.2, we discuss the details of resources used for conducting these experiments. Finally, Section 5.3 and 5.4 give the experimental results and analysis based on comparison of the algorithms mentioned.

5.1 Datasets

The experimental analysis is carried out on six real-world datasets (as shown in Table 5.1). We downloaded these datasets from the Laboratory of Web Algorithms. We symmetrized them and removed any self-loops to obtain simple undirected graphs. In our implementation, we store the graphs in a compressed web-graph format [9].

Essentially, the web-graph framework [9] enables us to load data in-memory part by part. Using this tool and about 10 GB memory allocated to the algorithms, we were able to process dewiki and ljournal on our proposed algorithm.

The sampled sub-graphs (induced by sets S and T) are stored in an adjacency hash-table using FASTUTIL[1] Java library, where each vertex v is associated with a hash-set containing its neighbors. The adjacency hash-table allows fast iterations over the neighbors of vertex v.

[1] FASTUTIL is a powerful Java package which provides faster and optimized versions of default Java collections such as maps, lists, sets, etc.

Dataset	Nodes	Edges	$d_{\max}$	4-cliques
enron	69,244	254,449	1,634	5,001,773
cnr	325,557	2,738,969	18,236	159,814,399
dblp	986,324	3,353,618	979	40,910,658
amazon	735,323	3,523,472	1,077	4,192,682
dewiki	1,532,354	33,093,029	118,246	158,337,013
ljournal	5,363,260	49,514,271	19,432	16,129,080,442

Table 5.1: Summary of real-world graphs. Here, $d_{\max}$ is the effective maximum degree.

The range of edge densities of the datasets used for experiments vary from 250K edges to 50M edges. Section 5.3 also shows how our algorithm, 4CDS, scales on some of these massive datasets such as *dewiki* and *ljournal*.

We provide a brief summary of the datasets [8, 10] used in this study:

1. **enron**: It consists of email communications between employees at Enron corporation. The dataset is made available by Federal Energy Regulatory.

2. **cnr-2000**: A small dataset crawled from an Italian domain called CNR (Consiglio Nazionale delle Ricerche).

3. **dblp-2011**: In this dataset, each node represents a scientist and an edge between two nodes depicts a collaboration on an article between two scientists.

4. **amazon**: This dataset, reported by the Amazon store, represents the similarities among books.

5. **dewiki-2013**: It depicts a snapshot of the German segment of Wikipedia (https://de.wikipedia.org/wiki/Wikipedia:Hauptseite).

6. **ljournal-2008**: denoted by **ljournal**, this dataset represents a social network (from a socail site) where nodes are users and an edge between these nodes is a sign of friendship between users. (https://www.livejournal.com/).

5.2 Experimental Settings

We implemented our code in Java 8 using the Webgraph library [9] for graph streams. This library compresses the graphs which helps in loading a graph in memory and also saves memory resources.[2] We used a standard intel core i5 machine equipped with 2.50 GHz processor and 8 GB of RAM. For larger datasets (dewiki and ljournal), we used Xeon E5620 processor as follows:

- Processor: Intel$^{(R)}$ Xeon$^{(R)}$ CPU E5620 @ 2.40 GHz
  Dual processors for a total of 16 threads
- Installed RAM: 64 GB
- Operating System: 64-bit Ubuntu 14.04.1 LTS

The default settings of JVM were not changed, implying that the heap configuration remained one-fourth of the system's total available memory. To conclude, the maximum available memory for the processing of our algorithm is 16 GB.

The comparison between 4CDS and the state-of-the-art algorithms is done considering the speedup of 4CDS over its competitors and the runtime of these algorithms. Such a comparison criteria is established to obtain a fair analysis. For robustness, we perform *ten* trials using distinct random seeds.

Parameter selection. In the 4CDS implementation, we can choose two parameters: the number of colors which can be assigned to a vertex and the sampling ratio that determines the number of triangle samples. In practice, we saw that setting the count of colors to 5 provided good estimates for all our datasets.

Evaluation metrics. The experiments measure the key performance metrics of accuracy (in terms of percentage error) and running time as shown in Table 5.2. The accuracy of our algorithm is measured (in terms of %) using formula $|C - X|/C$ (the lower the better).

5.3 Results

The key performance metrics, *i.e.* Accuracy and Running time, of our algorithm (details provided in Chapter 4.4) are measured by the experiments performed in this

[2]Some other works that have successfully used Webgraph for scaling various algorithms to big graphs are [12, 17, 25, 33, 41, 42].

Graph	T_{base}	T_{4CDS}	Speedup	Error (%)
enron	12.15	1.81	6.7	1.91
cnr	6.9K	2.40	176	3.78
dblp	22.42	4.56	4.9	3.43
amazon	9.36	2.61	3.6	1.65
dewiki	37K	91.14	400	1.63
ljournal	18K	141.24	130	0.78

Table 5.2: Experimental results for Runtime (in seconds), Accuracy (in percentage) and Speedups of 4CDS

section as shown in Table 5.2. Section 5.3.1 presents the accuracy analysis. In this section, we show how the estimation error of 4CDS varies with the sampling rate r. Section 5.3.2 presents runtime experimental results of 4CDS and analyze how the execution time of the proposed algorithm varies with r. Note that the baseline algorithm is used only to obtain the exact counts in order to calculate the relative errors. We don't employ the baseline algorithm in our algorithm.

5.3.1 Accuracy

In Table 5.2 we see in the last column the relative errors for each dataset when $c = 5$ and $r = 0.3$. The relative errors are all quite small. They vary from 0.78% to 3.78%. Figure 5.1 shows the relative errors computed for different settings of parameter r. The errors decrease fast as the sampling rate r goes up. From this figure, we can conclude that for dense graph networks such as *dewiki* and *cnr*, we get accurate results only when a sufficient number of triangles are sampled.

In addition, we vary input parameter c when employing color coding technique as shown in table 5.3. By assigning a random color in $[1, ..., c]$ to each node, we sparsify a graph stream. To arrive at the value for c, we tune the input parameter

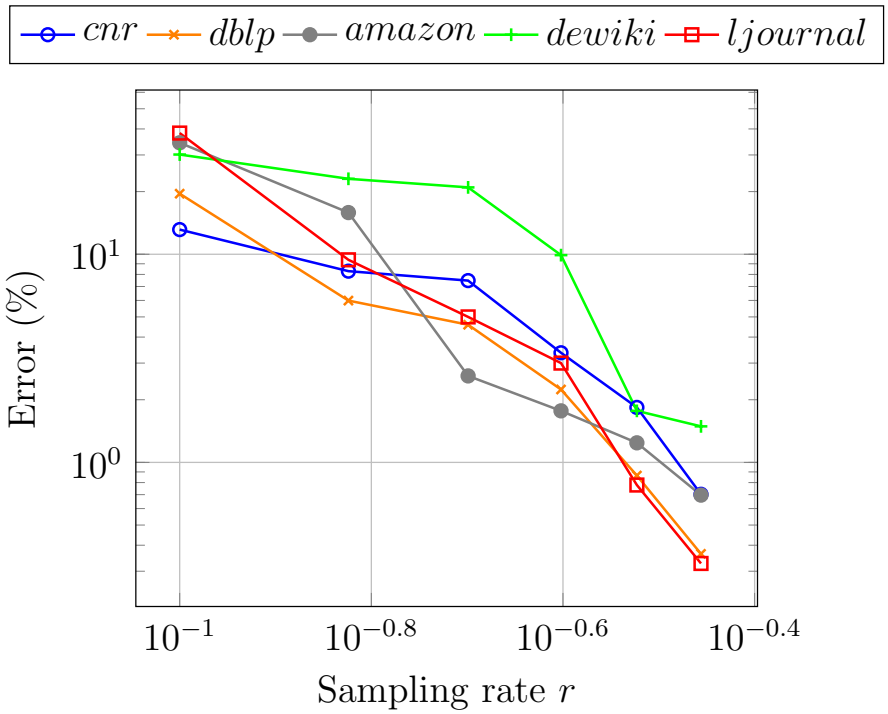


Figure 5.1: Accuracy of 4CDS for different r values ($c = 5$).

c by performing various experiments as follows. For evaluation purpose, we perform accuracy analysis by setting c equal to 3, 4 and 5. From table 5.3, we can see that the error rate increases as the value of c increases. The reason behind this is that as c value increases, the number of sampled edges decrease. This leads to higher error rate due to loss of information which occurs because of few sampled edges preserved in memory.

5.3.2 Runtime

In Table 5.2, we present the execution time results of our algorithm. Note that T_{base}, the baseline time required to perform an exact counting by enumerating all 4-cliques, does not only depend on the size of the graph, but also on the numbers of triangles. Figure 5.2 plots the running time against the sampling rate r on various datasets. We vary sampling rate r from 0.10 and increment it by a value of 0.05 till 0.35. Hence, we ran our experiment by sampling 10%, 15%, 20%, 30% and 35% of the triangles induced by sampled edges S.

Interestingly, while obtaining exact counting from baseline approach. *dewiki* requires longer run-time than *ljournal*. Even though *dewiki* is smaller by an order

Graph	$c = 4$	$c = 5$	$c = 6$
enron	1.43	2.22	3.88
cnr	0.70	1.99	2.13
dblp	0.32	0.86	2.37
amazon	0.70	1.24	2.60
dewiki	1.49	1.77	3.56
ljournal	0.46	0.78	3.31

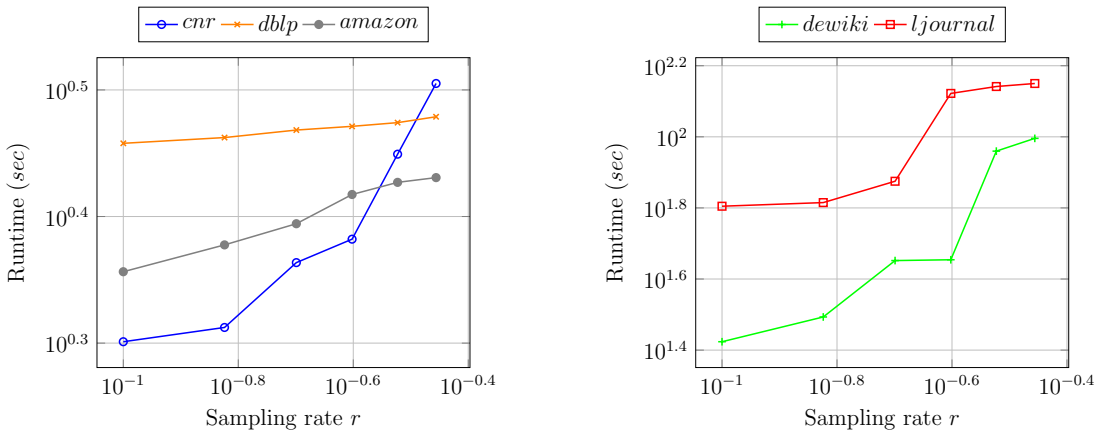
Table 5.3: Error(%) of 4CDS for different c values at $r = 0.30$

of magnitude, it is denser and has more 4-cliques. Our algorithm, 4CDS, handles *dewiki* effectively. From the plots in Figure 5.2, we can see that the algorithm processes sparse graphs like *dblp* and *amazon* much faster as compared to dense graphs such as *cnr* and *dewiki*. The *amazon* dataset, which has relatively small maximum degree can be processed in only a few seconds. Moreover, after a certain sampling rate, the runtimes for sparse graphs do not vary much unlike *cnr* and *dewiki*. The *dewiki* dataset has an enormous number of wedges (open triangles) and large maximum degree and so the baseline approach took about 10.5 hours to finish the enumeration. However, our 4CDS algorithm took merely a couple of minutes to estimate the global 4-clique count with more than 98% efficacy.

Another interesting analysis from Figure 5.2 especially in case of graphs such as *dewiki*, *dblp*, etc. is provided as follows. After scanning through initial K triangles, in 4CDS, the number of triangles do not continue to grow. The reason being the limit set on the reservoir size, which leads to a constant time required to iterate over all the sampled triangles. Hence, our algorithm tends to be faster than its competitors.

5.3.3 Scalability

Figure 5.3 depicts how the resource demand evolves with increasing workloads. The key objective behind measuring scalability is to examine if our algorithm can process infinite graph stream and not just medium size graphs. 4CDS scales well in graphs



(a) Runtime on *cnr*, *dblp* and *amazon* datasets

(b) Runtime on *dewiki* and *ljournal* datasets

Figure 5.2: Running time of 4CDS for different r values ($c = 5$).

billions of edges, even in densest graphs like *dewiki*. Hence, our algorithm achieves scalability without giving up quality.

5.4 Comparison with State-of-the-art

In this section, we compare our algorithm, 4CDS, with competing algorithms such as Color Coding [2], Edge Sampling [47], Graft [35] and Turán-Shadow [22] (detailed in 3). The rationale for this choice of algorithms is because they all share the idea of randomly sampling sets of edges using a reservoir. Still we recall that these algorithms do not work in a fully streaming setting, which is in contrast to our 4CDS algorithm.

Section 5.4.1 compares the accuracy for obtaining estimation counts of 4-cliques among the algorithms for various datasets. The objective of Section 5.4.2 is to compare the computation time of 4CDS with the existing methods. Note that Graft is infeasible for graphs with more than 10M edges. Hence, we could not finish executions on *dewiki* and *ljournal* for Graft.

5.4.1 Accuracy

In terms of accuracy, *Color Coding* and *Edge Sampling* gave the worst estimates. For *edge sampling*, we set the sampling ratio value initially to 0.1 and then increment it

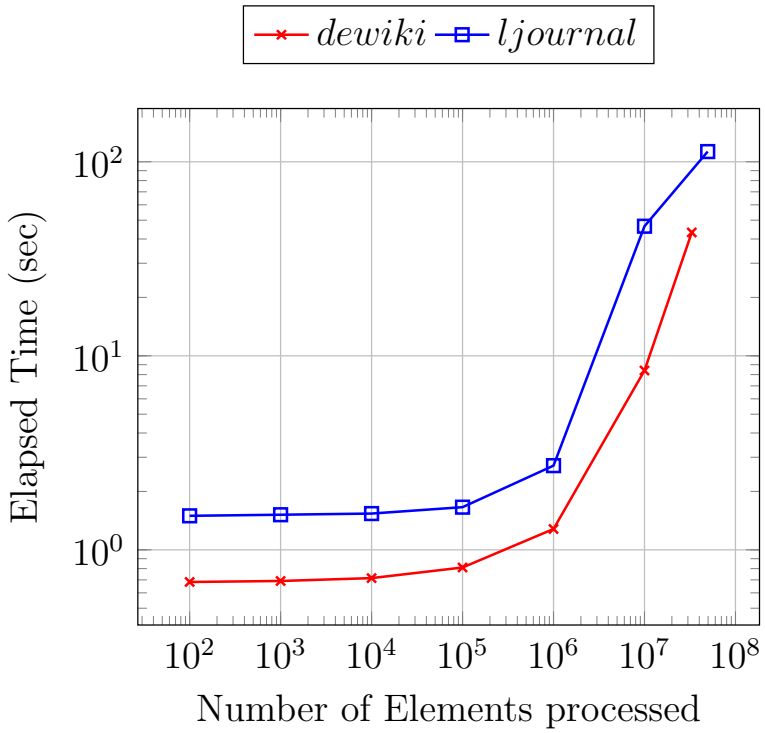


Figure 5.3: Scalability Analysis of 4CDS

by 0.15 up to 0.40. We observe that even though *Edge Sampling* is faster, it has very poor accuracy (especially for $r \leq 0.4$ in case of larger datasets).

The comparison results are shown in Figure 5.4. In terms of accuracy, we can see that Turán-Shadow and 4CDS are very close to each other. Even though Turán-Shadow performed slightly better on graphs such as *dblp* and *dewiki*, however, 4CDS wasn't much far behind. Our algorithm was still able to achieve the accuracy of less than 2% while simultaneously achieving large speedups and gains in terms of time as shown in Figure 5.5.

5.4.2 Running time

The comparison results are shown in Figure 5.5. The experiment shows that our algorithm is much faster than its competitors. More specifically, 4CDS is faster about 2× (on average) than its competitors. Our analysis shows that for dense graph networks like *cnr*, GRAFT has the worst execution time, however, it does provide competitive results on sparse graphs. In addition, Figure 5.5 also demonstrates that for massive networks like *dewiki* and *ljournal*, 4CDS is much faster than its competitors with

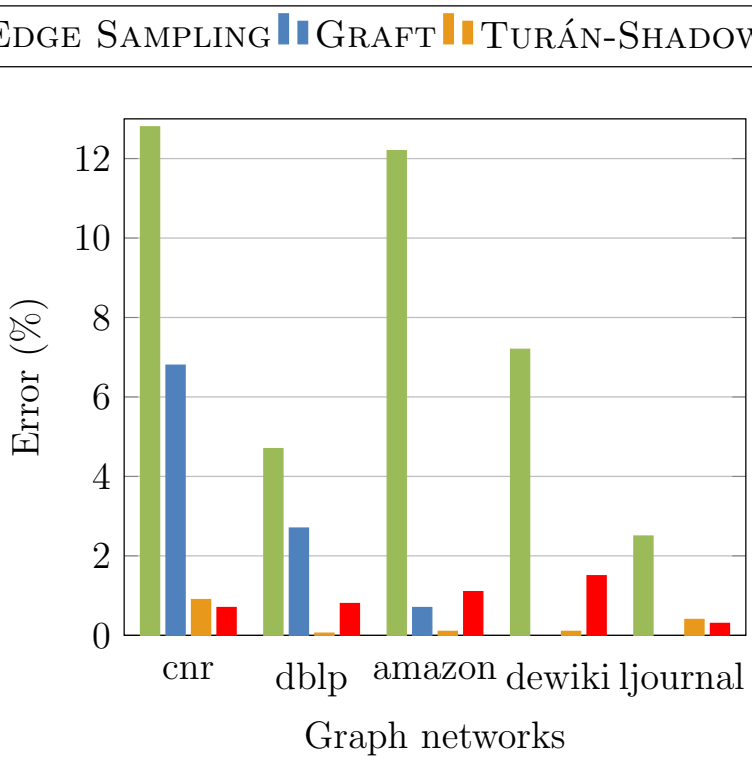


Figure 5.4: Accuracy Analysis for Edge Sampling, Graft, Turán-Shadow and 4CDS on various datasets

comparable accuracies.

Comparison of Graft and 4CDS: We are orders of magnitude faster when compared to Graft. The reason behind this is two-fold and is discussed as follows.

Firstly, Graft uses random sampling with a sampling rate r to sample edges from a graph G. This way it does a first scan through the graph. Secondly, it iterates over these sampled edges to compute the number of k-cliques that each of these preserved edges are a part of. And, for each sampled edge e_r, the embeddings grow by extending from any of the two vertices of e_r. Therefore, there can be $3 \times max[d(u), d(v)]$ possible ways on average for to obtain a 4-clique for a single sampled edge e_r. This makes the computation too extensive as the number of sampled edges increase, especially in case of large graphs.

Especially for dense graphs like *cnr* as shown in figure 5.5, we can see that Graft takes much longer time as compared to *dblp* even though *dblp* has larger number of edges.

4CDS, on the other hand, employs reservoir sampling rather than random sampling. This helps us to maintain a memory reservoir to store sampled triangles.

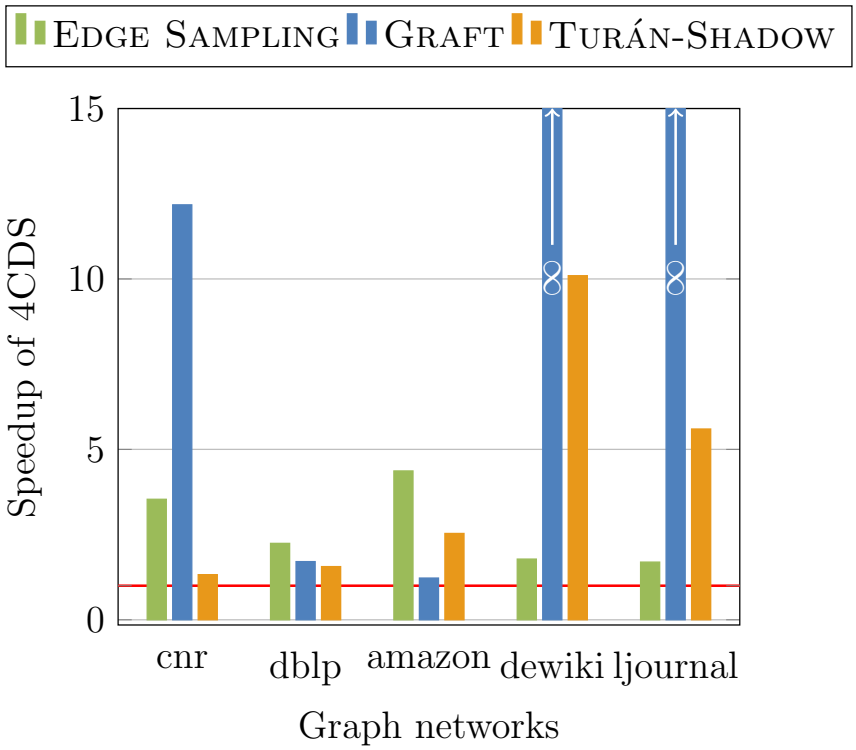


Figure 5.5: Speedup comparison of 4CDS over existing methods

Hence, it does not put any constraint on system's resources and the computations never extend to beyond a certain threshold. Due to this reason, 4CDS processes a graph stream within a reasonable time as depicted by Figure 5.5.

Comparison of Turán-Shadow and 4CDS: Among our algorithm and the most recent Turán-Shadow, our algorithm outperformed the latter considering the running time, but there is a trade-off of accuracy. Turán-Shadow is more accurate whereas 4CDS is much faster than Turán-Shadow.

Conclusion. Our algorithm, 4CDS, has better execution time when compared to existing methods. 4CDS outperformed all algorithms including recently proposed Turán-Shadow. However, Turán-Shadow proves to be more accurate, although, it consumes more space to store Turán Shadow structure.

Chapter 6

Conclusion

In this research work, we presented a randomized approach for efficiently approximating 4-cliques in a one-pass streaming model. The existing methods were too slow and did not scale well for higher order cliques. As the first step towards a one-pass non-parallel streaming approach, we presented 4CDS (4-Clique Dual Sampler), a randomized algorithm which uses dual sampling techniques to estimate 4-cliques. We demonstrated through extensive experiments using six real-world graphs that the estimates had smaller errors than those obtained by its state-of-the-art competitors.

Our algorithm randomly samples both edges and triangles to provide accurate estimates. On the arrival of an edge in a graph stream, the algorithm first counts the number of 4-cliques completed by the incoming edge in the subgraph induced by sampled triangles. Hence, the edges are utilized irrespective of whether they are sampled or discarded. The count obtained is then scaled up by the discovery probability of a 4-clique during the process. Once the counting procedure is completed, the edge is sampled using the color-coding technique, which assigns random colors to the edge endpoints and samples it if the color is a match. Once an edge is sampled, the triangles completed by it are sampled using reservoir sampling with a fixed memory budget. In addition, we theoretically and empirically prove that 4CDS maintains *unbiased* clique estimates. We also provide analysis for the complexity and accuracy of the algorithm.

The experiments were performed on a single commodity machine and 4CDS consistently outperformed its competitors in terms of speed. 4CDS achieves significant speedups without trading off much accuracy. For example, a real-world graph of 50M edges which takes around 1 GB can be processed in ≈ 150 seconds on a 4-core machine. Based on the experiments, we can conclude that our algorithm is about $2\times$

faster (on average) than its competitors. We were able to process massive graphs consisting of millions of edges and provide estimates for more than a billion 4-cliques in a single run within a reasonable amount of time.

Further, our experimental results also show that 4CDS gives significant savings in space compared to state-of-the-art and is scalable on large real-world datasets. Existing methods such as Graft [35] couldn't process graphs consisting of more than 10M edges. Hence, they couldn't scale to large graphs. The recently proposed Turán-Shadow [22] had its limitations as it stores an entire set of small subgraphs in-memory. This introduced a significant constraint on the system resources as the size of these subgraphs became larger than the graph itself (in the case of large datasets).

Over the years, the clique counting problem has been fundamental to many graph applications. There is a continuous trend to explore and incorporate details of bigger patterns, and we hope that the solutions provided as part of this research work lead to faster algorithms and provides useful insights for tasks performed on real-world networks.

Chapter 7

Future Work

This chapter presents the future work that can be done based on the contributions of this research work. The information presented in this chapter provides a clear research path for researchers who want to explore clique counting in a streaming setting.

There are numerous possible directions for future research. One possibility is to extend our result to five vertex or more high-ordered cliques. Our algorithm, 4CDS, only provides estimates for global 4-clique counts. Hence, another direction of this work can be to estimate per-vertex localized k-cliques counts.

Naturally, it would also be interesting to see if 4CDS can be employed when there are both edge insertions and deletions. In other words, our algorithm provides 4-clique estimates in an insertion-only graph streams as part of this research work. Hence, one might ask if it can handle a dynamic stream as well.

A few more natural extensions to 4CDS do come to mind as well. Firstly, we have directly implemented the theoretical algorithm in a streaming model on a single commodity machine without any parallelism. It is likely that we can achieve a faster heuristic if our algorithm can be parallelized. Hence, it would be beneficial to explore the feasibility of our algorithm in a MapReduce setting. This can definitely lead to significant improvements.

Moreover, while sampling edges our algorithm sparsifies a graph using color coding technique. An intriguing question is that whether our algorithm can achieve fine results while having a memory budget while sampling edges. If yes, then this can help in reducing the memory footprint of the algorithm.

It would also be interesting to extend our approach to probabilistic graphs where each edge contains a probability of existence, e.g. [18]. Incorporating edge probabili-

ties into sampling is worth exploring.

Another oblique approach would be to apply 4CDS to detect dense subgraphs which are formed in a short span of time. In real-world datasets, dense subgraphs have a certain characteristic to signal fraudulent behaviors or transactions. Some examples of such transactions include services for boosting followers on social media, manipulation of search engine rankings by websites, etc. One general framework in which dense subgraph discovery is heavily used is called correlation mining. In this case, the input dataset is processed into a graph by creating one vertex per entity. An edge between two entities represents the high correlation between them. Such type of mining is useful in various scenarios such as stock time series, gene expression profiles, etc.

From an alternate perspective if we let go of the random ordering of the vertices in a graph stream. Then, in the case of specific edge orderings such as endpoint groupings of edges, edges sorted by weights, etc.; one might be interested in exploring the complexity of several graph problems. All the above mentioned issues can be studied in future.

Bibliography

[1] Noga Alon, Phuong Dao, Iman Hajirasouliha, Fereydoun Hormozdiari, and S Cenk Sahinalp. Biomolecular network motif counting and discovery by color coding. *Bioinformatics*, 24(13):i241–i249, 2008.

[2] Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.

[3] Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997.

[4] Ziv Bar-Yossef, Ravi 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.

[5] Luca Becchetti, Paolo Boldi, Carlos Castillo, and Aristides 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.

[6] N. Betzler, R. van Bevern, M. R. Fellows, C. Komusiewicz, and R. Niedermeier. Parameterized algorithmics for finding connected motifs in biological networks. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 8(5):1296–1308, 2011.

[7] Mansurul A Bhuiyan, Mahmudur Rahman, Mahmuda Rahman, and Mohammad Al Hasan. Guise: Uniform sampling of graphlets for large graph analysis. In *2012 IEEE 12th International Conference on Data Mining*, pages 91–100. IEEE, 2012.

[8] Paolo Boldi, Marco Rosa, Massimo Santini, and Sebastiano Vigna. Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks. In Sadagopan Srinivasan, Krithi Ramamritham, Arun Kumar, M. P. Ravindra, Elisa Bertino, and Ravi Kumar, editors, *Proceedings of the 20th international conference on World Wide Web*, pages 587–596. ACM Press, 2011.

[9] Paolo Boldi and Sebastiano Vigna. The WebGraph framework I: Compression techniques. In *Proc. of the Thirteenth International World Wide Web Conference (WWW 2004)*, pages 595–601, Manhattan, USA, 2004. ACM Press.

[10] Paolo Boldi and Sebastiano Vigna. The WebGraph framework I: Compression techniques. In *Proc. of the Thirteenth International World Wide Web Conference (WWW 2004)*, pages 595–601, Manhattan, USA, 2004. ACM Press.

[11] Ilaria Bordino, Debora Donato, Aristides Gionis, and Stefano Leonardi. Mining large networks with subgraph counting. pages 737–742, 12 2008.

[12] Shu Chen, Ran Wei, Diana Popova, and Alex Thomo. Efficient computation of importance based communities in web-scale networks using a single machine. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 1553–1562. ACM, 2016.

[13] Norishige Chiba and Takao Nishizeki. Arboricity and subgraph listing algorithms. *SIAM Journal on computing*, 14(1):210–223, 1985.

[14] Seshadhri Comandur, Ali Pinar, and T. Kolda. Fast triangle counting through wedge sampling. *ArXiv*, abs/1202.5230, 2012.

[15] Maximilien Danisch, Oana Balalau, and Mauro Sozio. Listing k-cliques in sparse real-world graphs. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*, pages 589–598. International World Wide Web Conferences Steering Committee, 2018.

[16] Ethan R. Elenberg, Karthikeyan Shanmugam, Michael Borokhovich, and Alexandros G. Dimakis. Distributed estimation of graph 4-profiles. *CoRR*, abs/1510.02215, 2015.

[17] Fatemeh Esfahani, Venkatesh Srinivasan, Alex Thomo, and Kui Wu. Efficient computation of probabilistic core decomposition at web-scale. In *Advances in*

Database Technology-EDBT 2019, 22nd International Conference on Extending Database Technology, pages 325–336, 2019.

[18] Fatemeh Esfahani, Jian Wu, Venkatesh Srinivasan, Alex Thomo, and Kui Wu. Fast truss decomposition in large-scale probabilistic graphs. In *Advances in Database Technology-EDBT 2019, 22nd International Conference on Extending Database Technology*, pages 722–725, 2019.

[19] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826, 2002.

[20] Royi Itzhack, Yelena Mogilevski, and Yoram Louzoun. An optimal algorithm for counting network motifs. *Physica A: Statistical Mechanics and its Applications*, 381:482–490, 07 2007.

[21] Anand Padmanabha Iyer, Zaoxing Liu, Xin Jin, Shivaram Venkataraman, Vladimir Braverman, and Ion Stoica. Asap: Fast, approximate graph pattern mining at scale. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 745–761, 2018.

[22] S. Jain and Seshadhri Comandur. A fast and provable method for estimating clique counts using turán's theorem. *WWW'17*, 2017.

[23] Daniel Kane, Kurt Mehlhorn, Thomas Sauerwald, and He Sun. Counting arbitrary subgraphs in data streams. volume 7392, pages 598–609, 07 2012.

[24] Nadav Kashtan, Shalev Itzkovitz, Ron Milo, and Uri Alon. Efficient sampling algorithm for estimating subgraph concentrations and detecting network motifs. *Bioinformatics*, 20(11):1746–1758, 2004.

[25] Wissam Khaouid, Marina Barsky, Venkatesh Srinivasan, and Alex Thomo. K-core decomposition of large networks on a single pc. *Proceedings of the VLDB Endowment*, 9(1):13–23, 2015.

[26] Konstantin Kutzkov and Rasmus Pagh. Triangle counting in dynamic graph streams. In R. Ravi and Inge Li Gørtz, editors, *Algorithm Theory – SWAT 2014*, pages 306–318, Cham, 2014. Springer International Publishing.

[27] Ron Milo, Shai Shen-Orr, Shalev Itzkovitz, Nadav Kashtan, Dmitri Chklovskii, and Uri Alon. Network motifs: simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002.

[28] Rasmus Pagh and Charalampos E Tsourakakis. Colorful triangle counting and a mapreduce implementation. *Information Processing Letters*, 112(7):277–281, 2012.

[29] Gergely Palla, Imre Derényi, Illés Farkas, and Tamás Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 435(7043):814–818, Jun 2005.

[30] A. Pavan, Kanat Tangwongsan, Srikanta Tirthapura, and Kun-Lung Wu. Counting and sampling triangles from a graph stream. *Proc. VLDB Endow.*, 6(14):1870–1881, September 2013.

[31] Ali Pinar, Seshadhri Comandur, and Madhav Jha. From the birthday paradox to a practical sublinear space streaming algorithm for triangle counting.

[32] Ali Pinar, C Seshadhri, and Vaidyanathan Vishal. Escape: Efficiently counting all 5-vertex subgraphs. In *Proceedings of the 26th International Conference on World Wide Web*, pages 1431–1440. International World Wide Web Conferences Steering Committee, 2017.

[33] Diana Popova, Naoto Ohsaka, Ken-ichi Kawarabayashi, and Alex Thomo. Nosingles: a space-efficient algorithm for influence maximization. In *Proceedings of the 30th International Conference on Scientific and Statistical Database Management*, page 18. ACM, 2018.

[34] Nataša Pržulj, Derek G Corneil, and Igor Jurisica. Modeling interactome: scale-free or geometric? *Bioinformatics*, 20(18):3508–3515, 2004.

[35] Mahmudur Rahman, Mansurul Alam Bhuiyan, and Mohammad Al Hasan. Graft: An efficient graphlet counting method for large graph analysis. *IEEE Transactions on Knowledge and Data Engineering*, 26(10):2466–2478, 2014.

[36] Yudi Santoso, Venkatesh Srinivasan, and Alex Thomo. Efficient enumeration of four node graphlets at trillion-scale. In *Advances in Database Technology-EDBT 2020, 23rd International Conference on Extending Database Technology*, pages 439–442, 2020.

[37] Ahmet Erdem Sariyuce, C. Seshadhri, Ali Pinar, and Umit Catalyurek. Finding the hierarchy of dense subgraphs using nucleus decompositions. 11 2014.

[38] Ahmet Erdem Sariyuce, C Seshadhri, Ali Pinar, and Umit V Catalyurek. Finding the hierarchy of dense subgraphs using nucleus decompositions. In *WWW*, pages 927–937, 2015.

[39] Thomas Schank and D. Wagner. Approximating clustering coefficient and transitivity. *J. Graph Algorithms Appl.*, 9:265–275, 2005.

[40] Kijung Shin, Sejoon Oh, Jisu Kim, Bryan Hooi, and Christos Faloutsos. Fast, accurate and provable triangle counting in fully dynamic graph streams. *TKDD'20*, 14, 02 2020.

[41] M. Simpson, V. Srinivasan, and A. Thomo. Clearing contamination in large networks. *IEEE Transactions on Knowledge and Data Engineering*, 28(6):1435–1448, June 2016.

[42] Michael Simpson, Venkatesh Srinivasan, and Alex Thomo. Efficient computation of feedback arc set at web-scale. *Proceedings of the VLDB Endowment*, 10(3):133–144, 2016.

[43] Lorenzo De Stefani, Alessandro Epasto, Matteo Riondato, and Eli 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):43, 2017.

[44] Siddharth Suri and Sergei Vassilvitskii. Counting triangles and the curse of the last reducer. pages 607–614, 01 2011.

[45] Carlos HC Teixeira, Alexandre J Fonseca, Marco Serafini, Georgos Siganos, Mohammed J Zaki, and Ashraf Aboulnaga. Arabesque: a system for distributed graph mining. In *Proceedings of the 25th Symposium on Operating Systems Principles*, pages 425–440. ACM, 2015.

[46] Charalampos Tsourakakis. The k-clique densest subgraph problem. pages 1122–1132, 05 2015.

[47] Charalampos E Tsourakakis, U Kang, Gary L Miller, and Christos Faloutsos. Doulion: counting triangles in massive graphs with a coin. In *Proceedings of the*

15th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 837–846, 2009.

[48] Paul Turán. On an external problem in graph theory. *Mat. Fiz. Lapok*, 48:436–452, 1941.

[49] Jeffrey S Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)*, 11(1):37–57, 1985.

[50] Elisabeth Wong, Brittany Baur, Saad Quader, and Chun-Hsi Huang. Biological network motif detection: principles and practice. *Briefings in bioinformatics*, 13(2):202–215, 2012.

[51] Hao Yin, Austin R. Benson, and Jure Leskovec. Higher-order clustering in networks. *Physical Review E*, 97(5), May 2018.

[52] Zhao Zhao, Guanying Wang, Ali R Butt, Maleq Khan, VS Anil Kumar, and Madhav V Marathe. Sahad: Subgraph analysis in massive networks using hadoop. In *2012 IEEE 26th International Parallel and Distributed Processing Symposium*, pages 390–401. IEEE, 2012.

Appendix A

Performance Results of 4CDS

In this section, We provide the full results of the experiments performed for 4CDS algorithm. Below is the table that shows the computational running times and the error rates for all the graphs. The triangles are sampled from the subgraph induced by the sampled edges stored in-memory (as described in section 4.4).

Graphs	Speedup	Runtime(sec)	Error rate (%)
cnr	$> 2K$	2.31	7.47
dblp	7.62	2.94	4.58
amazon	3.77	2.48	2.6
dewiki	824.78	44.86	20.95
ljournal	240.03	74.99	5.1

Table A.1: Performance results of various graphs for 4CDS while storing 20% of triangles in memory.

Graphs	Speedup	Runtime(sec)	Error rate (%)
cnr	$> 2K$	2.41	3.36
dblp	7.57	2.96	2.24
amazon	3.59	2.61	1.77
dewiki	820.58	45.09	9.90
ljournal	135.90	132.45	3.01

Table A.2: Performance results of various graphs for 4CDS while storing 25% of edges in memory.

Graphs	Speedup	Runtime(sec)	Error rate (%)
cnr	$> 2K$	2.81	1.84
dblp	7.52	2.98	0.86
amazon	3.51	2.67	1.24
dewiki	405.97	91.14	1.77
ljournal	129.95	138.51	0.78

Table A.3: Performance results of various graphs for 4CDS while storing 30% of edges in memory.

Graphs	Speedup	Runtime(sec)	Error rate (%)
cnr	$> 2K$	3.2	0.70
dblp	7.44	3.012	0.36
amazon	3.48	2.69	0.70
dewiki	373.75	98.99	1.49
ljournal	127.44	141.24	0.33

Table A.4: Performance results of various graphs for 4CDS while storing 35% of edges in memory.