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

Anmol Mann
University of Victoria, Canada
anmolmann@uvic.ca

Venkatesh Srinivasan
University of Victoria, Canada
srinivas@uvic.ca

Alex Thomo
University of Victoria, Canada
thomo@uvic.ca

***Abstract*—Clique counting is considered to be a challenging problem in graph mining. The reason is combinatorial explosion; even moderate graphs with a few million edges could have clique counts in the order of many billions. In this paper, we propose 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. Our algorithm performs well on massive graphs containing several billions of 4-cliques, and terminates within a reasonable amount of time.**

***Index Terms*—stream data analysis, clique counting, approximation, randomized algorithm**

I. Introduction

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. [1], [2], [3] make use of cliques 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 [2]). 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 since 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, it is believed that algorithms that enumerate cliques or compute clique counts cannot terminate in a reasonable time [4] for large graphs. The clique counting problem has been studied extensively [5], [6], [7]. Alon et al. [5] proposed a technique to count given-length cycles. Bordino et al. [7] extends triangle counting to subgraphs on three and four vertices in a three-pass streaming model. Tiered-sampling [8] combines sampling of arbitrary subgraphs to count 4-cliques and 5-cliques in the one-pass streaming model. Other solutions, such as Arabesque [9] and 4-PROF-DIST [10] use distributed platforms.

ASONAM '21, November 8–11, 2021, Virtual Event, Netherlands
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-9128-3/21/11/$15.00
https://doi.org/10.1145/3487351.3489471

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 [3], [11], [12].

When it comes to approximating the number of 4-cliques, to the best of our knowledge, none of the previous works (except [8]) can handle the one-pass streaming model. On the other hand, [8] considers the one-pass streaming model but has a high computational complexity for processing each arriving edge, thus rendering it practically noncompetitive. In this paper, we show that by leveraging a dual sampling of both edges and triangles we can achieve an accurate and scalable 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 scalable approach to work in a fully streaming setting producing the 4-clique count in only a *single* pass. Our contributions are as follows:

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.
2) We provide a detailed analysis and prove that our estimation is unbiased. Our algorithm works for any type of edge ordering in a graph.
3) 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.
4) 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 1 4CDS (4-CLIQUE DUAL SAMPLER)

```
Input: Graph stream $G$
 1: global variables: $cliqueCount \leftarrow 0, T \leftarrow \emptyset$
 2: local variables: $S \leftarrow \emptyset$, number of colors $c$
 3: for each edge $e = (u, v) \in G$ do
 4:     COUNT4CLIQUE$(u, v)$
 5:     $colorU \leftarrow random\_int(0, c)$
 6:     $colorV \leftarrow random\_int(0, c)$
 7:     if $colorU == colorV$ then
 8:         Insert edge $e$ in $S$
 9:         $uSet \leftarrow N(u, S)$               ▷ neighbors of $u$ in $S$
10:         $vSet \leftarrow N(v, S)$               ▷ neighbors of $v$ in $S$
11:         for each $t \in uSet \cap vSet$ do
12:             if $coin\_toss(r) ==$ "heads" then
13:                 Insert triangle $(u, v, t)$ in set $T$
```

II. RELATED WORK

Chiba and Nishizeki [13] is a seminal work on clique enumeration. They split graphs into several subgraphs by using degeneracy ordering and then recursively list cliques in these subgraphs. KCLIST is a parallel clique enumeration technique proposed in [14] which improves on [13] but still could not scale for large graphs. Moreover, the running time of KCLIST depends on the ordering of nodes.

Milo et al. [15] 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, including some we have already discussed.

For *k*-clique counting, randomized techniques such as edge sampling [16], [17] and color coding [18], [19], [20] have been extensively used. The recently proposed ordering-based algorithm TURÁN-SHADOW [21] for estimating k-cliques ($k \leq 10$) (in a non-streaming setting) provides accurate clique estimates for large graphs. Note that *none* of the above works employs a streaming model. Thus they cannot obtain a clique count at any particular instant in time.

Tiered-sampling [8] approach uses a one-pass setting with fixed memory size. However, it counts and samples triangles on each edge insertion, therefore, increasing the computation time by many folds. In other words, using the tiered-sampling approach, TS4C$_1$ [8] would take $O(|E| \cdot |S|)$ (where S= set of edges sampled) extra amount of time on average as compared to our algorithm, 4CDS, to count the number of 4-cliques in a graph stream. We were unable to run TS4C$_1$ to completion on our single machine configuration in reasonable time.

As stated earlier, we consider a fully streaming setting and are able to efficiently produce an approximate clique count at any time instant. Our algorithm employs dual sampling (sampling both edges and triangles) as such sampling techniques have been used with much success as demonstrated in [22], [23], [24]. Additionally, our algorithm does not take into account the ordering of nodes unlike [13], [14].

Algorithm 2 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:         $cliqueCount \leftarrow cliqueCount + (count * scale)$
```

III. PRELIMINARIES

We consider simple, undirected graphs $G = (V, E)$. 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, $n = |V|$ denote the number of vertices, and $m = |E|$ denote the number of edges. The set of neighbors of a vertex $v \in V$ is $N(v)$ and the degree is $deg(v) = |N(v)|$. A 4-clique is a subgraph of four vertices where each vertex is neighbor of every other vertex. A 4-clique has 6 edges and four triangles.
Sampling Methods. We use the color-coding technique introduced by Alon et al. [18]. In this method, each vertex v of G is assigned a color c_v, which is a random number in $[1, c]$. This technique sparsifies G by preserving an edge only if the colors of its two endpoints are the same.

For sampling triangles, our algorithm uses the technique of reservoir sampling (with sampling ratio r), which chooses a random sample, without replacement, of k triangles from the set of triangles on the preserved edges after the color coding.

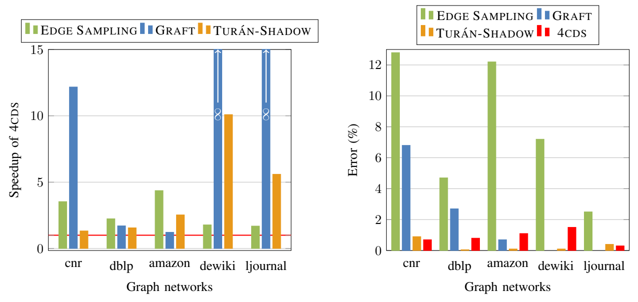
TABLE I: Datasets considered for the experiments

Dataset	n	m	$d_{\max}$	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

IV. THE ALGORITHM

Our algorithm for 4-clique approximation, denoted by 4CDS, is a sampling algorithm (Algorithm 1) that uses color coding and reservoir sampling.

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 2) 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



(a) Speedup (b) Accuracy Analysis

Fig. 1: Comparison of 4CDS with state-of-the-art algorithms EDGE SAMPLING, GRAFT, and TURÁN-SHADOW

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 2. 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 V below).

After counting the 4-cliques formed by edge e, we uniformly and randomly assign colors to both endpoints of e (lines 5-6). 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 uniformly at random using reservoir sampling with probability r (lines 12, 13).

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 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.

V. Analysis

Theorem V.1. 4CDS *(Algorithm 1) 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 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 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 $1/c^3$.

Regarding 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, $E[X] = C$. □

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

Graph	T_{base}	$T_{4\text{CDS}}$	*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

VI. Experiments

The real world datasets we use are listed in Table I. All the datasets were downloaded from the Laboratory for Web Algorithmics [25], [26], http://law.di.unimi.it/datasets.php. We symmetrized them and removed any self-loops to obtain simple undirected graphs. We implemented our code in Java using the Webgraph library [25] for graph streams. We used a standard intel core i5 machine equipped with 2.5 GHz processor and 8 GB of RAM. We notice, however, that the memory usage is less than 1 GB throughout the experiments except for the dewiki and ljournal graphs. For robustness, we perform *ten* trials using distinct random seeds.

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. We performed experimental analysis for different settings of input parameters. 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 metrics of accuracy (in terms of percentage error) and running time as shown in Table II. The accuracy of our algorithm is measured using formula $|C - X|/C$ (the lower, the better).

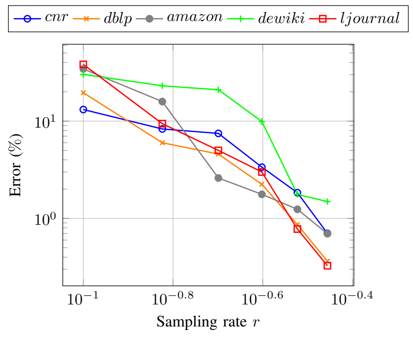


Fig. 2: Accuracy of 4CDS for different r values ($c = 5$).

Accuracy. In Table II we see in the last column the relative errors for each dataset when $c = 5$ and $r = 0.3$. The relative errors, w.r.t. exact counts, are all quite small. They vary from 0.78% to 3.78%. Figure 2 shows the relative errors computed for different settings of parameter r. The errors decrease fast as the sampling rate r goes up.

Runtime. In Table II, 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. Interestingly, *dewiki* requires longer run-time than *ljournal*. Even though *dewiki* is smaller by an order of magnitude, it is denser and has more 4-cliques. The *amazon* dataset, which has relatively small maximum degree can be processed in only a few seconds. The *dewiki* dataset has an enormous number of wedges (open triangles) and large maximum degree and so the enumeration algorithm 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.

Comparison with State-of-the-art. We compare 4CDS with the following algorithms: Color Coding [18], Edge Sampling [17], GRAFT [16] and TURÁN-SHADOW [21]. 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. GRAFT is infeasible for graphs with more than 10M edges. Therefore, we could not finish executions on *dewiki* and *ljournal* for GRAFT. For *edge sampling*, we set the sampling ratio value initially to 0.1 and then increment it by 0.15 up to 0.40. In terms of accuracy, *Color Coding* and *Edge Sampling* gave the worst estimates. We observe that even though *Edge Sampling* is faster, it has very poor accuracy. The comparison results are shown in Figure 1. 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 1 also demonstrates that for massive networks like *dewiki* and *ljournal*, 4CDS is much faster than its competitors with comparable accuracies.

VII. Conclusions

In this paper, we present a randomized approach for efficiently approximating 4-cliques in a one-pass streaming model. Our algorithm samples both edges and triangles to provide accurate estimates. 4CDS achieves significant speedups along with maintaining 96% accuracy. We are able to process massive graphs consisting of millions of edges and provide estimates for billions of 4-cliques in a single run within a reasonable amount of time. As future work, one direction is to extend our result to five vertex cliques. Another direction is to count per-vertex k-cliques.

References

[1] C. Tsourakakis, "The k-clique densest subgraph problem," in *WWW*, 2015, pp. 1122–1132.

[2] A. E. Sariyuce, C. Seshadhri, A. Pinar, and U. Catalyurek, "Finding the hierarchy of dense subgraphs using nucleus decompositions," in *WWW*, 2015, pp. 927–937.

[3] K. Shin, S. Oh, J. Kim, B. Hooi, and C. Faloutsos, "Fast, accurate and provable triangle counting in fully dynamic graph streams," *TKDD'20*, vol. 14, 02 2020.

[4] A. Pinar, C. Seshadhri, and V. Vishal, "Escape: Efficiently counting all 5-vertex subgraphs," in *WWW*, 2017, pp. 1431–1440.

[5] N. Alon, R. Yuster, and U. Zwick, "Finding and counting given length cycles," *Algorithmica*, vol. 17, no. 3, pp. 209–223, 1997.

[6] D. Kane, K. Mehlhorn, T. Sauerwald, and H. Sun, "Counting arbitrary subgraphs in data streams," in *ICALP*, 2012, pp. 598–609.

[7] I. Bordino, D. Donato, A. Gionis, and S. Leonardi, "Mining large networks with subgraph counting," in *ICDM*, 12 2008, pp. 737–742.

[8] L. D. Stefani, E. Terolli, and E. Upfal, "Tiered sampling: An efficient method for approximate counting sparse motifs in massive graph streams," *IEEE BigData 2017*, pp. 776–786, 2017.

[9] C. H. Teixeira, A. J. Fonseca, M. Serafini, G. Siganos, M. J. Zaki, and A. Aboulnaga, "Arabesque: a system for distributed graph mining," in *SOSP'15*. ACM, 2015, pp. 425–440.

[10] E. R. Elenberg, K. Shanmugam, M. Borokhovich, and A. G. Dimakis, "Distributed estimation of graph 4-profiles," *CoRR*, vol. abs/1510.02215, 2015. [Online]. Available: http://arxiv.org/abs/1510.02215

[11] L. D. Stefani, A. Epasto, M. Riondato, and E. Upfal, "Triest: Counting local and global triangles in fully dynamic streams with fixed memory size," *TKDD'17)*, vol. 11, no. 4, p. 43, 2017.

[12] A. P. Iyer, Z. Liu, X. Jin, S. Venkataraman, V. Braverman, and I. Stoica, "Asap: Fast, approximate graph pattern mining at scale," in *OSDI'18*, 2018, pp. 745–761.

[13] N. Chiba and T. Nishizeki, "Arboricity and subgraph listing algorithms," *SIAM Journal on computing*, vol. 14, no. 1, pp. 210–223, 1985.

[14] M. Danisch, O. Balalau, and M. Sozio, "Listing k-cliques in sparse real-world graphs," in *WWW'18*, 2018, pp. 589–598.

[15] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon, "Network motifs: simple building blocks of complex networks," *Science*, vol. 298, no. 5594, pp. 824–827, 2002.

[16] M. Rahman, M. A. Bhuiyan, and M. Al Hasan, "Graft: An efficient graphlet counting method for large graph analysis," *IEEE TKDE*, vol. 26, no. 10, pp. 2466–2478, 2014.

[17] C. E. Tsourakakis, U. Kang, G. L. Miller, and C. Faloutsos, "DOULION: counting triangles in massive graphs with a coin," in *KDD'09*, 2009, pp. 837–846. [Online]. Available: http://doi.acm.org/10.1145/1557019.1557111

[18] N. Alon, R. Yuster, and U. Zwick, "Color-coding," *J. ACM*, vol. 42, no. 4, pp. 844–856, 1995.

[19] Z. Zhao, G. Wang, A. R. Butt, M. Khan, V. A. Kumar, and M. V. Marathe, "Sahad: Subgraph analysis in massive networks using hadoop," in *In IPDPS'12*. IEEE, 2012, pp. 390–401.

[20] 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*, vol. 8, no. 5, pp. 1296–1308, 2011.

[21] S. Jain and S. Comandur, “A fast and provable method for estimating clique counts using turán’s theorem,” *WWW’17*, 2017.

[22] N. Alon, P. Dao, I. Hajirasouliha, F. Hormozdiari, and S. C. Sahinalp, “Biomolecular network motif counting and discovery by color coding,” *Bioinformatics*, vol. 24, no. 13, pp. i241–i249, 2008.

[23] R. Pagh and C. E. Tsourakakis, “Colorful triangle counting and a mapreduce implementation,” *Information Processing Letters*, vol. 112, no. 7, pp. 277–281, 2012.

[24] S. Comandur, A. Pinar, and T. Kolda, “Fast triangle counting through wedge sampling,” *ArXiv*, vol. abs/1202.5230, 2012.

[25] P. Boldi and S. Vigna, “The WebGraph framework I: Compression techniques,” in *WWW ’04*, 2004, pp. 595–601.

[26] P. Boldi, M. Rosa, M. Santini, and S. Vigna, “Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks,” in *WWW ’11*. ACM Press, 2011, pp. 587–596.