Computing Clustering Coefficients in Data Streams *

Lucian Salete Buriol † Gereon Frahling ‡ Stefano Leonardi §
Alberto Marchetti-Spaccamela ¶ Christian Sohler ‖

Abstract

We present random sampling algorithms that with probability at least $1 - \delta$ compute a $(1 \pm \epsilon)$-approximation of the clustering coefficient, the transitivity coefficient, and of the number of bipartite cliques in a graph given as a stream of edges. Our methods can be extended to approximately count the number of occurences of fixed constant-size subgraphs. Our algorithms only require one pass over the input stream and their storage space depends only on structural parameters of the graphs, the approximation guarantee, and the confidence probability. For example, the algorithms to compute the clustering and transitivity coefficient depend on that coefficient but not on the size of the graph. Since many large social networks have small clustering and transitivity coefficient, our algorithms use space independent of the size of the input for these graphs.

We implemented our algorithms and evaluated their performance on networks from different application domains. The sizes of the considered input graphs varied from about $8,000$ nodes and $40,000$ edges to about 135 million nodes and more than 1 billion edges. For both algorithms we run experiments with a sample set size varying from $100,000$ to $1,000,000$ to evaluate running time and approximation guarantee. Our algorithms appear to be time efficient for these sample sizes.

1 Introduction

The analysis of the structure of large networks often requires the computation of network indices based on counting the number of certain small subgraphs. In the analysis of complex networks, the clustering coefficient [21] is an important measure of the density of clusters in graphs and the degree at which clusters decompose in communities [5]. The clustering coefficient [21] is defined as the normalized sum of the fraction of neighbor pairs of a vertex of the graph that are connected. The related transitivity coefficient of a graph [9] is defined as the ratio between three times the number of triangles and the number of length two paths in the graph.

Frequent subgraphs in networks are also called *motifs*. Motifs are considered as the building blocks of universal classes of complex networks, whose detection sheds light in the process of network formation [19]. Specific motifs can be found with similar frequency in complex networks originated from the same application domain, as for instance in biochemistry, neurobiology, ecology, and engineering [18].

In more recent times, much attention has been devoted to the analysis of complex networks arising in information systems, from software systems, to overlay networks and physical connections. In the domain of Web applications, the observation of certain dense subgraphs of small size has been considered in the attempt of tracing the emergence of hidden cyber-communities. [12, 15]. For instance, a model of the process of growth of the hyperlinked structure of the Web [13], denoted as the *copying model*, use these dense subgraphs as building blocks of the process of formation of the webgraph.

*This work was partially supported by the EU within the 6th Framework Programme under contract 001907 “Dynamically Evolving, Large Scale Information Systems” (DELIS)

†Part of this work was done while the author was post-doc at Universitá degli Studi di Roma ”La Sapienza”

‡Heinz Nixdorf Institut – University of Paderborn

§Universitá degli Studi di Roma ”La Sapienza” and Carnegie mellon University

¶Universitá degli Studi di Roma ”La Sapienza”

‖Rutgers University and University of Paderborn

Counting the number of certain subgraphs in a large graph is a challenging computational task. The current state of the art provides methods that are either computational unfeasible on large data sets or do not provide any guarantee on the accuracy of the estimation. The best known methods for the solution of the simplest non trivial version of this problem, i.e. counting the number of triangles in a subgraph, reduces to matrix multiplication [4]. This is not computational feasible even on graphs of medium size, because of time complexity and the space required to store the whole graph and the related data structures in main memory. Schank and Wagner [20] give an extensive experimental study of the performance of algorithms for counting and listing triangles in graphs and computing the clustering coefficient. The state of the art on this subject is allso discussed in [14].

A natural way to address the problem of computing with massive data sets is to resort to the data stream model [10, 17]. In this model data arrives in a stream, one item at a time, and the algorithms are required to use very little space and per-item processing time. Secondary and slower memory storage devices naturally produce data streams for which multiple passes of computation are usually prohibitive due to the volumes of stored data. In several network contexts, the application receive data without pace from remote sources. Data stream computation allows also to compute on-line relevant quantities without incurring a large cost for organizing and storing data. We refer for instance to distributed crawlers collecting Web pages and their links, and performing structural analysis of the Webgraph prior to transfer data to a storage device.

Data stream algorithms have been proposed for problems like computation of frequency moments [1], histograms [8], Wavelet transforms [7], and others. This large body of work contrasts with a lack of efficient solutions of natural graph problems in the streaming model of computation [10]. Bar-Yosseff, Kumar and Sivakumar [22] give a first solution for counting triangles in the data stream model. They consider both the ”adjacency stream model” where the graph is presented as a sequence of edges in arbitrary order and there is no bound on the degree of a vertex, and the ”incidence stream” model where they consider only bounded-degree graphs and all edges incident to a vertex are presented successively. Their algorithms provide an ϵ approximation with probability $1 - \delta$ using a number of memory cells in some cases smaller than a naive sampling technique algorithm. The algorithms are obtained through a so called ”list” efficient reduction to the problem of computing frequency moments [1]. Subsequently, more algorithms have also been developed for the adjacency stream model [11].

Our results. In this paper we report on a stream of research aimed to develop random sampling data stream algorithms for computing network indices on very large graphs. In particular we concentrate on the clustering coefficient, the transitivity coefficient, and the number of bipartite cliques in a graph in the incidence stream model. Our algorithms, that find applications to the problems of detecting the existence of dense clusters in a graph, are based on the random sampling data stream algorithms that approximate the number of triangles in a graph in the adjacency stream and the incidence stream model presented in [2, 3].

To compute the transitivity coefficient it essentially suffices to compute the number of triangles in a graph. We develop a data structure for this task that uses $O(\frac{1}{\epsilon^2}\log(\frac{1}{\delta})\log(|V|)(1+\frac{|T_2|}{|T_3|}))$ memory cells, where T_i denotes the set of node-triples having i edges in the induced subgraph. This improves by a quadratic factor the result of [22]. Observe that $\frac{|T_3|}{|T_2|}$ is exactly equal to $1/3$ of the inverse of the *transitivity coefficient* of the graph, an universal measure whose value for networks of practical interest is hardly bigger than 10^5.

We also present a 1-pass streaming algorithm which with probability $1-\delta$ returns a $(1\pm\epsilon)$-approximation on the clustering coefficient C_G of a graph G when the graph is given as a incidence stream. It needs $\mathcal{O}(\log\frac{\log|V|}{\delta}\cdot\frac{1}{\epsilon^2\cdot C_G})$ memory cells.

Denote by $K_{i,j}$ the set of complete bipartite cliques in the graph where each of i vertices link to all of j vertices. As a further contribution we provide a data stream algorithm that provides an approximation of the number of $K_{3,3}$ of the graph in the incidence stream model ordered by destination nodes with outdegree bounded by Δ which needs $O\left(\log(|V|)\cdot\frac{|K_{3,1}|\cdot\Delta^2\ln(\frac{1}{\delta})}{|K_{3,3}|\cdot\epsilon^2}\right)$ memory cells.

We also provide an optimized implementation of the two pass version of the presented data stream algorithms and a test on networks including large webgraphs, graphs of the largest online encyclopedia Wikipedia [16], graphs of collaborations between actors and authors.

Our algorithm for approximating the transitivity coefficient provide excellent approximations with a sample of size 10^5. For the algorithm that estimates the number of bipartite cliques, we find out that a number of 10^5 samples already suffices to detect a large number of bipartite cliques. We expect similar or even better results for the algorithm that approximates the clustering coefficient that we plan to implement and test in the near future.

2 Preliminaries

We consider the following models of computation for undirected graphs in data streams. Let $G = (V, E)$ denote a directed graph without self-loops. We assume that G is given as a stream of incidence lists. Let $\mathcal{L}(v)$ denote the incidence list of vertex v. The incidence list of vertex v contains all edges that are directed to a vertex v, i.e. all edges $e \in E$ of the form $e = [u, v\rangle$ for some $u \in V$. When we consider undirected graphs, we simply assume that every edge is represented by two undirected edges.

Definition of Clustering Coefficient. Let $G = (V, E)$ be an undirected graph. For every vertex $v \in V$ let $\mathcal{N}(v)$ denote its neighborhood, i.e. $\mathcal{N}(v) = \{u \in V : \exists(u, v) \in E\}$. The *clustering coefficient* C_v of a vertex $v \in V$ of G is defined as the probability that a random pair of its neighbors is connected by an edge, i.e. $C_v := \frac{\left|\left\{(u,v)\in E : u\in\mathcal{N}(v) \text{ and } v\in\mathcal{N}(v)\right\}\right|}{\binom{|\mathcal{N}(v)|}{2}}$. In case of $|\mathcal{N}(v)| < 2$ we define $C_v := 0$.

The *clustering coefficient* C_G of G is the average clustering coefficient of its vertices, i.e. $C_G = \frac{1}{n} \cdot \sum_{v \in V} C_v$.

Transitivity Coefficient. The transitivity coefficient of a graph is defined as the ratio between three times the number of triangles in a graph devided by the number of paths of length 2. Since in our model it is easy to compute the number of paths of length 2, it suffices to compute the number of triangles in a graph to compute the transitivity coefficient.

3 Approximating the Clustering Coefficient

In this section we sketch how one can obtain a 1-pass algorithm to approximate the clustering coefficient. We start with the following algorithm from [20], which can be implemented as a 2-pass algorithm

```
ApproxClusteringCoefficient(G, s)
    sample s vertices w_1, ..., w_s uniformly at random
    for i = 1 to s do
        sample a random pair (u, v), u ≠ v, of points from N(w_i)
        if (u, v) ∈ E then X_i = 1
        else X_i = 0
    Output X := 1/s · Σ_{i=1}^{s} X_i
```

It is easy to see that the algorithm can be implemented in two passes over the data. One pass to selected the random vertices and the random pairs of neighbors and another pass to check for each pair of neighbors whether they are connected by an edge. The next corollary follows immediately from [20][Theorem 1].

Corollary 3.1 *There is a 2-pass streaming algorithm which with probability* $1 - \delta$ *returns a* $(1 \pm \epsilon)$*-approximation on the clustering coefficient* C_G *of a graph* G *when the graph is given as a incidence stream. It needs* $\mathcal{O}(\log \frac{1}{\delta} \cdot \frac{1}{\epsilon^2 \cdot C_G})$ *memory cells.*

3.1 A one-pass algorithm

We show that it is also possible to get a one-pass algorithm. Again we sample a vertex w uniformly at random, pick two of its neighbors uniformly at random, and check whether these neighbors are connected by an edge.

To pick two random neighbors of w we use random hash functions in a way somewhat similar to random sampling in dynamic data streams [6]. We will require $\log n$ guesses 2^j for the degree of w. For each guess we pick a random hash function $h_j : V \to \{1, ..., 2^j\}$. For the right value of j the hash function will map with constant probability exactly two vertices from the neighborhood $\mathcal{N}(w)$ of w to the value 1, i.e. $|h^{-1}(1) \cap \mathcal{N}(w)| = 2$. Conditioned on this event, the two vertices are distributed uniformly at random among $\mathcal{N}(w)$. In this case the algorithm outputs a random variable X with expected value C_G, in all other cases it outputs an error ($\perp$). For simplicity of presentation, we assume fully random hash functions.

In the algorithm u_j, v_j are random variables for the first and second neighbor x of w with $h_j(x) = 1$. The variable X_j denotes the output value, if $j = \lceil \log d \rceil$, where d is the degree of w. If we do not have $|h^{-1}(1) \cap \mathcal{N}(w)| = 2$, we set $X_j = \perp$.

To implement the algorithm as a one pass streaming algorithm we have to care about the tests $w \in \mathcal{L}(x)$? and $u_j \in \mathcal{L}(x)$? that have to be performed by the algorithm. Since both w and u_j are known when we start to parse $\mathcal{L}(x)$ (or $u_j = \perp$, which means the second test is not executed) we can maintain this information on the fly.

```
OnePassClusteringCoefficient
  sample a vertex w uniformly at random
  for j = 1 to log V + 1 do
     X_j ← ⊥; u_j ← ⊥; v_j ← ⊥
     h_j ← random hash function h : V → {1, ..., 2^j}
  for each incidence list L(x) in the stream do
     for j = 1 to log V + 1 do
        if h_j(x) = 1 and w ∈ L(x) then              // (x ∈ N(w) will be sampled)
           if u_j = ⊥ then u_j ← x                   // (x is first random neighbor of w)
           else
              if v_j = ⊥ then
                 v_j = x                             // (x is second random neighbor of w)
                 if u_j ∈ L(x) then X_j ← 1          // (check, if there is edge between u_j and v_j)
                 else X_j ← 0
              else X_j ← ⊥                           // (|h^{-1} ∩ N(w)| > 2)
     if x = w then d ← |L(x)|                        // (set the degree of w to the right value)
  if d < 2 then output 0
  if d ≥ 2 then output X_⌈log d⌉
```

Theorem 1 *With probability $\frac{1}{44}$ the algorithm* OnePassClusteringCoefficient *does not output $\perp$. If it does not output $\perp$ it outputs a $0-1$ random variable X with expected value* $\mathbf{E}[X] = C_G$. □

To approximate the clustering coefficient in one pass we start $s = \Theta(\log \frac{1}{\delta} \cdot \frac{44}{\epsilon^2 \cdot C_G})$ instances. From Chernoff Bounds it follows that with probability at least $1 - \delta/2$ at least $\frac{\log(1/\delta)}{\epsilon^2 C_G}$ instances report a success. From the previous section it is clear that using the results of the successful instances the clustering coefficient can be approximated up to a multiplicative error of ϵ with probability $1 - \delta$.

Theorem 2 *There is a 1-pass streaming algorithm which with probability $1-\delta$ returns a $(1 \pm \epsilon)$-approximation of the clustering coefficient C_G of a graph G when the graph is given as an incidence stream. It uses $\mathcal{O}(\frac{\log(1/\delta) \cdot \log(|V|)}{\epsilon^2 C_G})$ memory cells.* □

4 Transitivity Coefficient

We will only describe a 3-pass algorithm. Using techniques of a similar flavour as in the previous section it is possible to combine these passes to a 1-pass algorithm. Since the algorithm also computes the number of paths of length 2, we immediately get an approximation for the transitivity coefficient.

SAMPLETRIANGLE

1st. Pass:

Count the number of paths of length 2 in the stream.

2nd. Pass:

Uniformly choose one of these paths using UNIFORMTWOPATH of [3]

Let (a, v, b) be this path

3rd. Pass:

Test if edge (a, b) appears within the stream.

if $(a, b) \in E$ **then** $\beta = 1$

else $\beta = 0$

return β

The number of paths of length 2 is exactly $P := |T_2| + 3 \cdot |T_3| = \sum_{i=1}^{|V|} d_i \cdot (d_i - 1)/2$. Thus we can count the number of paths of length 2 by computing the degree of each vertex. The second pass can be implemented using reservoir sampling. The algorithm SAMPLETRIANGLE outputs a value β with expected value $\mathbf{E}[\beta] = \frac{3 \cdot |T_3|}{|T_2| + 3 \cdot |T_3|}$. Using similar techniques as in the previous section we can achieve high concentration and combine the three paths into a single pass. We summarize our results in the following theorem.

Theorem 3 *There is a 1-Pass streaming algorithm to count the number of triangles in incidence streams up to a multiplicative error of* $1 \pm \epsilon$ *with probability at least* $1 - \delta$*, which needs* $O(s \cdot \log |V|)$ *memory cells and amortized expected update time* $O(\log(|V|) \cdot (1 + s \cdot (\frac{|V|}{|E|})))$*, where*

$$s \geq \frac{3}{\epsilon^2} \cdot \frac{|T_2| + 3 \cdot |T_3|}{|T_3|} \ln(\frac{2}{\delta}) \ .$$

5 Counting $K_{3,3}$

Using similar but somewhat more involved techniques as in the previous sections we can also estimate the number of $K_{3,3}$ in a directed graph.

Theorem 4 *There is a 1-Pass streaming algorithm to count the number of* $K_{3,3}$ *in incidence streams ordered by destination nodes with outdegree bounded by* Δ *up to a multiplicative error of* ϵ *with probability at least* $1 - \delta$*, which needs* $O\left(\log(|V|) \cdot \frac{K_{3,1} \cdot \Delta^2 \ln(\frac{1}{\delta})}{K_{3,3} \cdot \epsilon^2}\right)$ *memory cells.* □

6 Computational Experiments

We run our experiments on three datasets. The first dataset consists of an instance of the `webgraph` of 135 million nodes and 1 billion edges. It was obtained from a graph extracted in 2001 by the WebBase project at Stanford [23].

The second data set contains instances used in the experiments reported in [20]. These instances include two social networks based on coplay of actors, one network based on coauthorship in computer science, an instance based on the 2002 Google contest, and a network of internet routers and their connections. The third set of instances is originated from the link structure of Wikipedia [16], from an old dump of June 13, 2004 [16].

We give some typical experimental results for the problem of computing the transitivity coefficient, i.e. the problem of counting the number of triangles in a graph. For all runs of all instances we detected one or more triangles in the sample used. The average percentage deviation is rather small. Even for a sample size of 1,000 samples we can get reasonable good results. The average percentage deviation from the exact number of triangles for all instances, but the `webgraph`, are 5.10%, 2.17% and 0.85% for the sample sizes of 10,000, 100,000 and 1,000,000, respectively. Similar results are obtained for the problem of counting the number of bipartite cliques and are also expected for the implementation of the algorithm for approximating the clustering coefficient.

Graph	r=10,000			r=100,000			r=1,000,000			$\frac{2T_3}{T_2+3T_3}$
	$\tilde{T_3}$	Qlt(%)	Time	$\tilde{T_3}$	Qlt(%)	Time	$\tilde{T_3}$	Qlt(%)	Time	
webgraph	7,991,057,264	-	153.78	7,541,370,749	-	393.78	7,993,479,298	-	490.56	
	6,461,924,928	-	153.55	7,384,193,673	-	392.20	8,097,287,808	-	490.00	
	9,977,868,646	-	153.69	8,337,706,066	-	393.92	7,591,170,489	-	491.28	
actor2004	1,127,610,593	-4.16	12.29	1,155,564,261	-1.79	33.28	1,181,693,982	0.43	35.84	0.174932
	1,111,095,851	-5.57	12.52	1,192,599,566	1.36	20.28	1,177,782,402	0.10	35.11	
	1,177,449,181	0.07	12.12	1,175,270,762	-0.11	20.30	1,178,307,250	0.14	85.48	
google-2002	43,353	-1.22	0.28	45,489	3.65	1.20	44,765	2.00	4.97	0.004922
	45,293	3.20	0.28	45,435	3.52	1.00	43,704	-0.42	4.85	
	37,346	-14.91	0.27	42,420	-3.34	0.99	44,208	0.73	7.55	
actor2002	344,973,896	-0.53	6.70	345,817,151	-0.29	11.93	347,151,238	0.10	24.36	0.110693
	351,507,109	1.35	6.59	347,683,085	0.25	12.03	345,810,766	-0.29	24.38	
	330,775,554	-4.62	6.62	344,359,433	-0.71	12.00	347,532,178	0.21	55.16	
authors	1,636,611	-1.73	0.43	1,665,394	-0.01	1.21	1,670,148	0.28	4.47	0.227631
	1,586,971	-4.71	0.44	1,648,484	-1.02	1.19	1,665,792	0.02	4.45	
	1,633,188	-1.94	0.44	1,650,487	-0.90	1.20	1,664,291	-0.07	6.86	
itdk0304	458,517	0.76	0.33	449,558	-1.21	1.24	457,604	0.56	4.58	0.040506
	399,317	-12.25	0.34	458,260	0.70	1.11	451,481	-0.79	4.44	
	438,002	-3.75	0.34	453,440	-0.36	1.11	451,358	-0.81	6.40	

References

[1] Noga Alon, Yossi Matias, and Mario Szegedy, *The space complexity of approximating the frequency moments*, 1996, pp. 20–29.

[2] L. Buriol, G. Frahling, S. Leonardi, A. Marchetti-Spaccamela, and C. Sohler, *Counting graph minors in data streams*, 2005, DELIS technical report, http://delis.upb.de/paper/DELIS-TR-0245.pdf.

[3] ______, *Counting triangles in data streams*, To appear in Proceedings of the ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 06, 2006.

[4] Don Coppersmith and Shmuel Winograd, *Matrix multiplication via arith- metic progressions*, Journal of Symbolic Computation **3**, no. 9.

[5] Stephen Eubank, V. S. Anil Kumar, Madhav V. Marathe, Aravind Srinivasan, and Nan Wang, *Structural and algorithmic aspects of massive social networks*, SODA '04: Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms (Philadelphia, PA, USA), Society for Industrial and Applied Mathematics, 2004, pp. 718–727.

[6] C. Sohler G. Frahling, P. Indyk, *Sampling in dynamic data streams and applications*, 21st Annual Symposium on Computational Geometry (2005).

[7] Anna C. Gilbert, Yannis Kotidis, S. Muthukrishnan, and Martin Strauss, *Surfing wavelets on streams: One-pass summaries for approximate aggregate queries.*, VLDB, 2001, pp. 79–88.

[8] Sudipto Guha, Nick Koudas, and Kyuseok Shim, *Data-streams and histograms*, ACM Symposium on Theory of Computing, 2001, pp. 471–475.

[9] Frank Harary and Helene J. Kommel, *Matrix measures for transitivity and balance*, Journal of Mathematical Sociology **6**, 199210.

[10] M. Henzinger, P. Raghavan, and S. Rajagopalan, *Computing on data streams*, 1998.

[11] Hossein Jowhari and Mohammad Ghodsi, *New streaming algorithms for counting triangles in graphs.*, COCOON, 2005, pp. 710–716.

[12] R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins, *Trawling the web for emerging cyber communities*, (1999), 403–416.

[13] Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, D. Sivakumar, Andrew Tomkins, and Eli Upfal, *Random graph models for the web graph.*, FOCS, 2000, pp. 57–65.

[14] M. Latapy, *Theory and practice of triangle problems in very large (sparse (power-law)) graphs*, 2006.

[15] L. Laura, S. Leonardi, S. Millozzi, and J.F. Sybeyn, *Algorithms and experiments for the webgraph*, Proc. European Symposium on Algorithms (ESA), 2002.

[16] S. Leonardi S. Millozzi L.S. Buriol, D. Donato, *Link and temporal analysis of wikigraphs*, Technical Report (2005).

[17] S. Muthukrishnan, *Computing on data streams*, 2005.

[18] N. Kashtan R. Levitt S. Shen-Orr I. Ayzenshtat M. Sheffer R. Milo, S. Itzkovitz and U. Alon, *Superfamilies of designed and evolved networks*, Science **303** (2004), 1538–42.

[19] Shalev Itzkovitz Nadav Kashtan Dmitri Chklovskii Ron Milo, Shai Shen-Orr and Uri Alon, *Network motifs: Simple building blocks of com- plex networks*, Science **298**, no. 509, 824 – 827.

[20] Thomas Schank and Dorothea Wagner, *Approximating clustering coefficient and transitivity*, Journal of Graph Algorithms and Applications **9** (2005), no. 2, 265–275.

[21] Duncan J. Watts and Steven H. Strogatz, *Collective dynamics of small- world networks*, Nature **393**, 440–442.

[22] D. Sivakumar Z. Bar-Yosseff, R. Kumar, *Reductions in streaming algorithms, with an application to counting triangles in graphs*, Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms (2002), 623–632.

[23] J. Zhao, *An implementation of min-wise independent permutation family*, (2005), http://www.icsi.berkeley.edu/ zhao/minwise/.