

Estimating Clustering Indexes in Data Streams*

Luciana S. Buriol
Instituto de Informática
Universidade Federal
do Rio Grande do Sul, Brazil
(buriol@inf.ufrgs.br)

Gereon Frahling
Google Research
New York, NY
(gereon@google.com)

Stefano Leonardi
Dipartimento di Informatica e Sistemistica
Università di Roma “La Sapienza”, Italy
(Stefano.Leonardi@dis.uniroma1.it)

Christian Sohler
Heinz Nixdorf Institute and
Dept. of Computer Science
University of Paderborn
(csohler@upb.de)

Abstract

We present random sampling algorithms that with probability at least $1 - \delta$ compute a $(1 \pm \epsilon)$ -approximation of the clustering coefficient and of the number of bipartite clique subgraphs of a graph given as an incidence stream of edges. The space used by our algorithm to estimate the clustering coefficient is inversely related to the clustering coefficient of the network itself. The space used by our algorithm to compute the number $K_{3,3}$ of bipartite cliques is proportional to the ratio between the number of $K_{1,3}$ and $K_{3,3}$ in the graph.

Since the space complexity depends only on the structure of the input graph and not on the number of nodes, our algorithms scale very well with increasing graph size. Therefore they provide a basic tool to analyze the structure of dense clusters in large graphs and have many applications in the discovery of web communities, the analysis of the structure of large social networks and the probing of frequent patterns in large graphs.

We implemented both algorithms and evaluated their performance on networks from different application domains and of different size; The largest instance is a webgraph consisting of more than 135 million nodes and 1 billion edges. Both algorithms compute accurate results in reasonable time on the tested instances.

1 Introduction

The analysis of the structure of large networks often requires the computation of network indexes based on the number of certain small subgraphs. Much attention has recently been devoted to the structural analysis of networks arising in information systems; Physical telecommunication connections, overlay networks, and software systems are some examples. The observation of certain dense subgraphs in the webgraph, the graph formed by web pages and hyperlinked connections, has also been considered in the attempt of tracing the emergence of hidden cyber-communities [12]. The counted subgraphs are typically dense bipartite cliques of small size that are interpreted as cores of web communities: The vertices in the left partition of the bipartite clique are considered as member pages pointing to a set of centers/authorities for the community. A large number of these subgraphs has been observed in large crawls of the Web by Kumar et al. [12] and by Laura et al. [14]. A stochastic model of graphs that resembles the structure of the Web [13], called the *copying model*, uses these dense subgraphs as building blocks of the process of network

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

formation of the Webgraph. Counting the number of such subgraphs therefore provides deeper insight into the construction process of large graphs and can provide justifications for different construction models.

Another network index widely used in the analysis of complex network structures is the *clustering coefficient* [18]. It is defined as the normalized sum of the fraction of neighbor pairs of a vertex that are connected. It measures the degree at which clusters decompose into communities [6]. See Section 3 for an exact definition of the clustering coefficient.

Estimating the value of network indexes in large graphs is a challenging computational task. Current state of the art methods are either computationally unfeasible on large data sets or do not provide guarantees 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 graph, use matrix multiplication [4]. Even on graphs of medium size this is not computationally feasible because of both the time complexity and the main memory usage required to store the whole graph. Schank and Wagner [17] gave an extensive experimental study of algorithms counting and listing triangles and computing the clustering coefficient in small and medium size graphs. Their algorithms work in main memory and report results for graphs up to 668 thousand nodes.

A natural way to address massive data sets consisting of more than 100 million nodes is the data stream model [10, 15]. 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. Data stream algorithms are able to sequentially read the data from secondary memory storage devices and avoid slow random access to the data. Even massive data sets which do not fit on any available storage device could be handled.

Data stream algorithms have been proposed for various problems. Examples are the computation of frequency moments [1], histograms [9], or Wavelet transforms [8]. The large body of work on data stream algorithms contrasts with a lack of efficient solutions of natural graph problems in the streaming model of computation [10]. Bar-Yossef, Kumar and Sivakumar [19] gave a first solution for counting triangles in the data stream model. They considered both the “adjacency stream model” where the graph is presented as a sequence of edges in arbitrary order 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 to count triangles have been developed for the adjacency stream model [11, 2], and the incidence stream model [2].

Schank and Wagner [17] gave an algorithm, which returns with probability $1 - \delta$ a $(1 \pm \epsilon)$ -approximation on the clustering coefficient C_G of a graph G when the graph is given as an incidence stream. It needs two passes over the data and $\mathcal{O}(\log \frac{1}{\delta} \cdot \frac{1}{\epsilon^2 \cdot C_G})$ memory cells. We will recap the algorithm in Section 4.

2 Our results

In this paper we present random sampling data stream algorithms to compute the clustering coefficient and the number of bipartite cliques in graphs given as an incidence stream.

We improve the 2-pass streaming algorithm of [17] to estimate the clustering coefficient C_G . Our new algorithm only requires one pass over the stream of edges. Although the memory needed by our improved algorithm is slightly larger ($\mathcal{O}(\log \frac{1}{\delta} \cdot \frac{\log |V|}{\epsilon^2 \cdot C_G})$ memory cells compared to $\mathcal{O}(\log \frac{1}{\delta} \cdot \frac{1}{\epsilon^2 \cdot C_G})$ memory cells needed by the 2-pass algorithm), our technique enables the application of the algorithm in real streaming scenarios. Distributed crawlers collecting Web pages and their links could perform structural analysis of the Webgraph online while transferring the data to storage devices. Our new algorithm is also applicable to graphs which are too large to be stored at all.

We also provide a data stream algorithm that computes a $(1 \pm \epsilon)$ - approximation of the number of $K_{3,3}$ (bipartite cliques having three nodes in each partition) in a graph given as an incidence stream ordered by destination nodes with outdegree bounded by Δ . The algorithm needs $\mathcal{O} \left(\log(|V|) \cdot \frac{|K_{3,3}| \cdot \Delta^2 \cdot \ln(\frac{1}{\delta})}{|K_{3,3}| \cdot \epsilon^2} \right)$ memory cells. Previous techniques to count subgraphs [11, 2] could only be applied to the estimation of subgraphs containing a star (triangles or complete cliques K_i for example). Our algorithm is based on a new sampling technique, which can be extended to estimate the number of bipartite cliques $K_{i,j}$ for arbitrary i

and j . Since the space complexities of our algorithms depend only on the structure of the input graph (and not on its size), they can be used to estimate these structural properties even for large webgraph crawls.

We present optimized implementations of the algorithms and tests on networks including large webgraphs, graphs of the largest online encyclopedia Wikipedia [3], and graphs of collaborations between actors and authors. Our results show that for all networks a relatively small number of memory cells already suffices to provide good approximations of the clustering coefficient and the number of bipartite cliques.

2.1 Structure of the paper

We present in Section 4 the algorithm to estimate the clustering coefficient and in Section 5 the algorithm to count $K_{3,3}$ cliques. Section 6 introduces the optimized implementations of the algorithms which are tested on real data sets in Section 7.

3 Preliminaries

Let $G = (V, E)$ denote a directed graph without self-loops. We assume that G is given as a stream of incidence lists, one incidence list $\mathcal{L}(v)$ for each node $v \in V$. The incidence list $\mathcal{L}(v)$ of a node v consists of all edges that are directed to v , i.e. all edges $e \in E$ of the form $e = [u, v]$ for some $u \in V$. The incidence lists can appear in arbitrary order in the stream. We also do not assume a particular order of the edges of an incidence list. When we consider undirected graphs, we simply assume that every edge is represented by two undirected edges, appearing in the incidence lists of both nodes.

There are natural motivations for the model of incidence lists. Big graphs as the webgraph are often retrieved by crawlers who essentially produce streams of incidence lists. Furthermore even a stream of all known edges in the webgraph can be transformed into a stream of incidence lists by one round of a parallel computing environment such as Google's MapReduce (the complete process is described in [5]).

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.

$$C_v := \frac{|\{(u, v) \in E : u \in \mathcal{N}(v) \text{ and } v \in \mathcal{N}(v)\}|}{\binom{|\mathcal{N}(v)|}{2}} .$$

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.

$$C_G = \frac{1}{n} \cdot \sum_{v \in V} C_v .$$

4 Approximating the Clustering Coefficient

In this section we recapitulate how to approximate the Clustering Coefficient using an algorithm from [17], which we state only for the unweighted case.

```

APPROXCLUSTERINGCOEFFICIENT( $G, s$ )
  sample  $s$  vertices  $w_1, \dots, w_s$  uniformly at random
  for  $i = 1$  to  $s$  do
    sample a random pair  $(u, v)$ ,  $u \neq v$ , of points uniformly from  $\mathcal{N}(w_i)$ 
    if  $(u, v) \in E$  then set  $X_i \leftarrow 1$ 
    else set  $X_i \leftarrow 0$ 
  Output  $X := \frac{1}{s} \cdot \sum_{i=1}^s X_i$ 

```

It is easy to see that the algorithm can be implemented in two passes over the data. One pass to select 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. For sake of completeness we give an analysis of the algorithm below. We first show that the expected value of X_i is exactly C_G . We have for each $i \in \{1, \dots, s\}$:

$$\mathbf{E}[X_i] = \frac{1}{n} \cdot \sum_{v \in V} \mathbf{E}[X_i \mid w_i = v] = \frac{1}{n} \cdot \sum_{v \in V} C_v = C_G .$$

Then we use the fact that for 0 – 1 random variables we have

$$\mathbf{Var}[X_i] \leq \mathbf{E}[X_i^2] = \mathbf{E}[X_i] = C_G .$$

Now we analyze the variance of X . Since the X_i are mutually independent we get

$$\mathbf{Var}[X] = \mathbf{Var}\left[\frac{1}{s} \cdot \sum_{i=1}^s X_i\right] = \frac{1}{s^2} \cdot \sum_{i=1}^s \mathbf{Var}[X_i] \leq \frac{C_G}{s} .$$

Finally, we can apply Chebyshev inequality. This gives us

$$\Pr[|X - \mathbf{E}[X]| \geq \epsilon \cdot \mathbf{E}[X]] \leq \frac{\mathbf{Var}[X]}{(\epsilon \cdot \mathbf{E}[X])^2} \leq \frac{C_G}{s \cdot \epsilon^2 \cdot C_G^2} = \frac{1}{s \cdot \epsilon^2 \cdot C_G} .$$

If $s \geq \frac{3}{\epsilon^2 \cdot C_G}$ then with probability $2/3$ the algorithm APPROXCLUSTERINGCOEFFICIENT approximates the clustering coefficient of G within a relative error of $(1 \pm \epsilon)$. A standard success amplification technique (running the algorithm $\Theta(\log \frac{1}{\delta})$ times and returning the median of all results) leads to the following corollary, which follows immediately from [17][Theorem 1].

Corollary 4.1 *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.*

Remark 4.2 *The memory we use depends directly on the clustering coefficient itself. Since we do not know the exact clustering coefficient, this can be seen as a drawback of the algorithm. To overcome this problem we provide in Appendix A a method which takes a memory bound s and a data stream as input and returns an approximation of the clustering coefficient together with an estimation of the accuracy. The description of the algorithm then no longer depends on the value of the clustering coefficient itself.*

Since the theoretical bounds for the method of Appendix A are rather loose, we suggest another approach for practical applications: Start our algorithms using all available memory cells, do a few parallel experiments and estimate the error by the deviation of the results of the different runs. Our implementation results show that one gets good approximations of the clustering coefficient using that approach.

4.1 A one-pass algorithm

In this section we show that it is possible to reduce both passes of the algorithm above to one pass over the incidence stream. 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 [7]. We will require a guess 2^j of the degree of w for each value of $j \in \{0, \dots, \lceil \log n \rceil\}$. For each guess we pick a random hash function $h_j : V \rightarrow \{1, \dots, 2^j\}$. Since we can count the degree of w while passing over the data stream, we can at the end of our algorithm pick the right value of j (having approximately $2^j = \text{degree}(w)$) and forget all other values of 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. $|h_j^{-1}(1) \cap \mathcal{N}(w)| = 2$. Conditioned on this event, these two vertices are distributed uniformly at random among $\mathcal{N}(w)$. They will be our two sampled vertices. The algorithm outputs a random variable X having expected value C_G . In case we do not have exactly two neighbors of w mapped to 1 by h , it outputs an error (\perp). 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 having $h_j(x) = 1$. The variable X_j denotes the output value for $j = \lceil \log d \rceil$, where d is the degree of w .

```

ONEPASSCLUSTERINGCOEFFICIENT
sample a vertex  $w$  uniformly at random
for  $j = 1$  to  $\lceil \log V \rceil$  do
   $X_j \leftarrow \perp$ ;  $u_j \leftarrow \perp$ ;  $v_j \leftarrow \perp$ 
   $h_j \leftarrow$  random hash function  $h : V \rightarrow \{1, \dots, 2^j\}$ 
for each incidence list  $\mathcal{L}(x)$  in the stream do
  for  $j = 1$  to  $\lceil \log V \rceil$  do
    if  $h_j(x) = 1$  and  $w \in \mathcal{L}(x)$  then // ( $x \in \mathcal{N}(w)$  will be sampled)
      if  $u_j = \perp$  then  $u_j \leftarrow x$  // ( $x$  is first sampled neighbor of  $w$ )
    else
      if  $v_j = \perp$  then
         $v_j = x$  // ( $x$  is second sampled neighbor of  $w$ )
        if  $u_j \in \mathcal{L}(x)$  then  $X_j \leftarrow 1$  // (check, if there is edge between  $u_j$  and  $v_j$ )
        else  $X_j \leftarrow 0$ 
      else  $X_j \leftarrow \perp$  // ( $|\mathcal{N}(w) \cap \mathcal{N}(x)| > 2$ )
  if  $x = w$  then  $d \leftarrow |\mathcal{L}(x)|$  // (set the degree of  $w$  to the right value)
  if  $d < 2$  then output  $0$ 
  if  $d \geq 2$  then output  $X_{\lceil \log d \rceil}$ 

```

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 having expected value $\mathbf{E}[X] = C_G$.*

Proof : Let d be the degree of node w and $\tilde{d} = 2^{\lceil \log d \rceil}$. We have $d \leq \tilde{d} \leq 2 \cdot d$. Let be $h := h_{\lceil \log d \rceil}$, $u := u_{\lceil \log d \rceil}$, and $v := v_{\lceil \log d \rceil}$. At the end the algorithm chooses the result $X_{\lceil \log d \rceil}$. The algorithm outputs $X \neq \perp$ iff exactly 2 nodes adjacent to w are hashed to 1 by h . In that case these two nodes are chosen uniformly at random. The first of these nodes is stored in variable u , the second in variable v . Finally $X_{\lceil \log d \rceil}$ is set to one iff the incidence list of v contains node u , so iff w, u , and v form a triangle.

It remains to show that with probability $1/44$ exactly two neighbouring nodes are hashed to 1 by h . There are d nodes adjacent to w , each one is hashed to 1 by h with probability $\frac{1}{\tilde{d}}$. Let x_1, \dots, x_d be the neighbouring nodes of w . If $d < 2$ the algorithm always outputs the correct value $X = 0$. For $d \geq 2$ we obtain:

$$\begin{aligned}
& \Pr[|\{x \in \mathcal{N}(w) | h(x) = 1\}| = 2] \\
&= \sum_{i=1}^{d-1} \frac{1}{\tilde{d}} \cdot \Pr[\forall x \in \{x_1, \dots, x_{i-1}\} h(x) \neq 1 \wedge |\{x \in \{x_{i+1}, \dots, x_d\} | h(x) = 1\}| = 1] \\
&= \sum_{i=1}^{d-1} \frac{1}{\tilde{d}} \cdot \left(1 - \frac{1}{\tilde{d}}\right)^{i-1} \cdot \sum_{j=i+1}^d \frac{1}{\tilde{d}} \cdot \left(1 - \frac{1}{\tilde{d}}\right)^{d-i-1} = \frac{1}{\tilde{d}^2} \left(1 - \frac{1}{\tilde{d}}\right)^{d-2} \cdot \sum_{i=1}^{d-1} (d-i) \\
&= \frac{1}{\tilde{d}^2} \cdot \frac{d(d-1)}{2} \cdot \left(1 - \frac{1}{\tilde{d}}\right)^{d-2} \geq \frac{d(d-1)}{8d^2} \cdot \left(1 - \frac{1}{\tilde{d}}\right)^d \geq \frac{1}{8} \left(1 - \frac{1}{\tilde{d}}\right)^d \cdot \frac{1}{e} \geq \frac{1}{16e} \geq \frac{1}{44}
\end{aligned}$$

□

To approximate the clustering coefficient in one pass we start $s \cdot r$ instances of ONEPASSCLUSTERINGCOEFFICIENT for $s = \Theta(\log \frac{1}{\delta} \cdot \frac{1}{\epsilon^2 \cdot C_G})$ and $r = \log \frac{2s}{\delta} / \log \frac{44}{43}$. We devide the instances into s groups of r instances.

Fix one group. We bound the probability that all instances in the group return \perp : For each instance the probability to return \perp is bounded by $43/44$, therefore we have a probability of at most $(\frac{43}{44})^r = \frac{\delta}{2s}$ for the event that all instances of the group return \perp . If this is the case for one of our groups, our algorithm fails. Since we have s groups, each of them failing with probability at most $\frac{\delta}{2s}$, our algorithm fails with probability at most $\frac{\delta}{2}$ by the union bound.

We now consider the case that all groups have at least one instance reporting a value 0 or 1. This case happens with probability at least $1 - \frac{\delta}{2}$. We take the result of the first non-failing instance of each

group. Then we have s independent $\{0, 1\}$ -variables, each one having expected value C_G . Therefore we can proceed exactly as in the two-pass case ($\Theta(\log \frac{1}{\delta})$ times averaging $\frac{3}{\epsilon^2 C_G}$ results and taking the median of these averaged results). As shown in the two-pass case this leads to a $(1 \pm \epsilon)$ -approximation with probability $1 - \delta$. We can also apply the technique of Appendix A to obtain an algorithm whose description does not depend on C_G and which outputs a guarantee of the multiplicative error of the estimation.

Theorem 2 *There is a 1-pass streaming algorithm which returns with probability $1 - \delta$ 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 $\mathcal{O}(\frac{\log(1/\delta) \cdot \log(|V|)}{\epsilon^2 C_G} \cdot \log \frac{1}{\epsilon \delta C_G})$ memory cells. \square*

5 Counting $K_{3,3}$

In this section we consider the problem to estimate the number of complete bipartite subgraphs $K_{3,3}$ in a directed graph G of bounded outdegree Δ . We assume that the graph is given as a stream of incidence lists ordered by destination nodes (as described in the Preliminaries). Let $K_{3,3}$ denote the number of $K_{3,3}$ subgraphs and $K_{3,1}$ denote the number of $K_{3,1}$ subgraphs. We first assume that we have a method to choose a $K_{3,1}$ uniformly at random.

$K_{3,3}$ COUNTING

```

Choose a  $K_{3,1}$  uniformly at random from all  $K_{3,1}$  in  $G$ .
Let the three edges of the  $K_{3,1}$  be  $[a, u]$ ,  $[b, u]$  and  $[c, u]$ 
Pass over the stream and stop when all 3 edges of the  $K_{3,1}$  are read
Select  $x_1, x_2$  uniformly at random from  $\{a, b, c\}$ 
Select  $k_1, k_2$  uniformly at random from  $\{1, 2, \dots, \Delta\}$ 
if  $k_1 = k_2 \wedge x_1 = x_2$  then output  $\beta = 0$ 
Continue to pass over the stream.
Select the  $k_1$ -th outgoing edge of  $x_1$  (counting from the stop). Call this edge  $[x_1, v]$ .
Select the  $k_2$ -th outgoing edge of  $x_2$  (counting from the stop). Call this edge  $[x_2, w]$ .
Once  $[x_1, v]$  has been selected: check, if  $[a, v], [b, v], [c, v]$  are present in the remaining stream
Once  $[x_2, w]$  has been selected: check, if  $[a, w], [b, w], [c, w]$  are present in the remaining stream
if both is the case then output  $\beta = 1$  else output  $\beta = 0$ .

```

Lemma 5.1 *Algorithm $K_{3,3}$ COUNTING outputs a random value β having expected value*

$$\mathbf{E}[\beta] = \frac{2 \cdot K_{3,3}}{9 \cdot \Delta^2 \cdot K_{3,1}}$$

Proof : Let $K = (A, B, C, U, V, W)$ be an arbitrary fixed $K_{3,3}$ with edges directed from A, B, C to U, V and W . Let X_K denote the indicator random variable for the event that $\beta = 1$ with witness K . We would like to determine the probability for $X_K = 1$. W.l.o.g., let U be the vertex being first within the incidence list, V, W occurring after U within the stream. Also, w.l.o.g. let us assume that $[A, V]$ is the first edge in the stream among the edges from $\{A, B, C\}$ to V and let $X \in \{A, B, C\}$ be the node such that $[X, W]$ is the first edge among the edges from $\{A, B, C\}$ to W in the stream. We have $X_K = 1$, if the following events occur:

- A, B, C, U are chosen as $K_{3,1}$ with U being the destination node (otherwise we can not detect the edges to U).
- edges $[A, V]$ and $[X, W]$ are selected by the algorithm (only then we have a chance to see the remaining edges of the $K_{3,3}$ in the remainder of the stream).

The probability of the first event is $1/|K_{3,1}|$. Conditioned on the first event the probability to choose $\{v, w\} = \{V, W\}$ is $2/\Delta^2$: each edge $[x_1, \cdot]$ appearing after $[x_1, U]$ in the stream has a probability of $1/\Delta$ to be chosen by the algorithm. We know that $[x_1, V]$ and $[x_1, W]$ appear after $[x_1, U]$ in the stream. Therefore each of these two edges has a probability of $1/\Delta$ to be chosen. By a similar argument we have

independently a probability of $1/\Delta$ to choose the edge $[x_2, V]$ resp. $[x_2, W]$. We select the nodes V and W if we either select $[x_1, V]$ and $[x_2, W]$ or $[x_1, W]$ and $[x_2, V]$. Therefore the probability to choose $\{v, w\} = \{V, W\}$ and w is $2/\Delta^2$.

Assume now that we have chosen A, B, C, U as the $K_{3,1}$ and the edges $[x_1, V]$ and $[x_2, W]$. This assumption is satisfied with probability $2/(\Delta^2 \cdot |K_{3,1}|)$, independent of the choice of x_1 and x_2 . The probability for $x_1 = A$ and $x_2 = X$ is $1/9$, since x_1 and x_2 are chosen uniformly from $\{A, B, C\}$. Therefore the probability to detect the $K_{3,3}$ and to set $X_K = 1$ is $2/(9 \cdot \Delta^2 \cdot |K_{3,1}|)$. Since we fixed the $K_{3,3}$, the probability to detect an arbitrary $K_{3,3}$ this way is then $\frac{2 \cdot |K_{3,3}|}{9 \cdot \Delta^2 \cdot |K_{3,1}|}$. \square

To complete the description of the algorithm, we show how we can choose a $K_{3,1}$ uniformly from the stream in the first step of our algorithm. This is done using a similar method as choosing the length-2-paths in an algorithm to count the number of triangles in a graph[2]. We start $O(\log n)$ different instances of the algorithm below in parallel. Each instance corresponds to a guess 2^j , $1 \leq j \leq 4 \log n$ of the number of $K_{3,1}$ in the graph. We also count the number $K_{3,1} = \sum_{i=1}^{|V|} d_i \cdot (d_i - 1) \cdot (d_i - 2)/6$. In the end we can choose an instance whose guess is at most a factor 2 away from the number of $K_{3,1}$. So from now on let us assume we know the number of $K_{3,1}$ (up to a factor 2). We will extend the method UNIFORMTWOPATH from [2]. Let $f(x) = \binom{x}{3}$. The algorithm is as follows.

```

UNIFORMK3,1
  Select value k uniformly from the set  $\{1, \dots, K_{3,1}\}$ .
  For each node v in the incidence stream do:
    If k > 0 then
      Set h  $\leftarrow \lceil f^{-1}(k) \rceil$ 
      Set k2  $\leftarrow k - f(h - 1)$ 
      Set i  $\leftarrow \left\lceil \sqrt{2k_2 + \frac{1}{4}} + \frac{1}{2} \right\rceil$ 
      Set j  $\leftarrow i - \frac{i^2 - i}{2} + k_2 - 1$ 
      Pass over the complete incidence list of node v.
      If incidence list of v contains more than j edges then
        a  $\leftarrow$  the hth node in the incidence list of v
        b  $\leftarrow$  the ith node in the incidence list of v
        c  $\leftarrow$  the jth node in the incidence list of v
        u  $\leftarrow$  v
      end if
      d  $\leftarrow$  degree of node v
      k  $\leftarrow k - \frac{d \cdot (d-1) \cdot (d-2)}{6}$ 
    end if
  end do
  return edges (a, u), (b, u) and (c, u)

```

The idea of the algorithm above is to enumerate the $K_{3,1}$'s that are incident to one node v in a way that all $K_{3,1}$ whose edges belong to the first i edges incident to v are enumerated before those using an edge which appears later in the stream. We select the k th $K_{3,1}$ with respect to that enumeration order. Given a number k we can compute a triple (h, i, j) telling us to select the h -th, i -th, and j -th edge incident to the current vertex. If the current vertex has less than j edges, we know that the node v is incident to less than k $K_{3,1}$ subgraphs. We simply subtract the number of $K_{3,1}$ choices for this vertex from k and start with our new k and the next vertex.

Equivalent to the algorithms of Section 4 we can run the algorithm $K_{3,3}$ COUNTING in parallel $\Theta(\log \frac{1}{\delta} \cdot \frac{\Delta^2}{\epsilon^2 \cdot K_{3,3}})$ times. We divide the instances into $\Theta(\log \frac{1}{\delta})$ groups of size $\frac{27 \cdot \Delta^2 \cdot K_{3,1}}{\epsilon^2 \cdot 2 \cdot K_{3,3}}$, take the average result of each group and return the median of these group values. The returned value is with probability $1 - \delta$ a $(1 \pm \epsilon)$ -approximation of the value $\frac{2 \cdot K_{3,3}}{9 \cdot \Delta^2 \cdot K_{3,1}}$. Since we can in parallel count the value $K_{3,1}$, a multiplication of the median with $\frac{9 \cdot \Delta^2 \cdot K_{3,1}}{2}$ will yield a $(1 \pm \epsilon)$ -approximation of the number of $K_{3,3}$ with probability $1 - \delta$. The technique of Appendix A to output an approximation guarantee can be applied as well.

Theorem 3 *There is a 1-pass streaming algorithm which returns with probability $1 - \delta$ a $(1 \pm \epsilon)$ -approximation of the number of $K_{3,3}$ subgraphs of a graph G when the graph is given as an incidence stream ordered by destination nodes with outdegree bounded by Δ . It uses $\mathcal{O}\left(\log(|V|) \cdot \log\left(\frac{1}{\delta}\right) \cdot \frac{K_{3,1} \cdot \Delta^2}{K_{3,3} \cdot \epsilon^2}\right)$ memory cells.* \square

6 Implementation of the Data Stream Algorithms

In this section we describe several optimization steps used in the implementation of the algorithms to count $K_{3,3}$'s and to estimate the clustering coefficient.

First, we use some memory blocks in order to improve I/O efficiency. Instead of reading single edges consecutively from the hard disk, a large amount of edges is read at once and stored in a main memory block of fixed size. This also allows to distinguish between processing time and reading time.

Our algorithms consist of s instances passing in parallel over the stream of edges. Instead of feeding each edge to each of the s instances we use optimized hashing approaches. Each instance describes a set of possible edges or nodes that it is interested in. These sets of interesting edges are then inserted into a hashtable. After reading an edge our algorithm can then quickly identify the different instances which are interested in that particular edge (resp. the end nodes of the edge).

We use simple hash functions to implement the hash table. For hashtables storing edges (i.e. two indices u and v) we use hash functions h of the form $h(u, v) = rnd_1 \cdot u + rnd_2 \cdot v \pmod{2s}$, where rnd_1 and rnd_2 are two randomly generated numbers between one and s (sample set size). For hashtables storing single nodes we use hash functions h of the form $h(u) = rnd_1 \cdot u \pmod{2s}$. The size of the hash tables is $2 \cdot s$. Our hashtables do not provide theoretical guarantees on the lookup time. We nevertheless chose them because they provide very fast hashing time in all our experiments.

The implementations of the one pass algorithm to estimate the clustering coefficient and the two pass algorithm to count $K_{3,3}$'s are described in the next subsections.

6.1 One Pass Algorithm to Estimate the Clustering Coefficient

To simplify the implementation we store the incidence list of a single node in main memory. The memory consumption for this operation is negligible, since all our input instances are sparse graphs.

The hashing function is implemented as described above. We chose this particular set of hash functions because they are extremely fast and provide good approximations (see results).

6.2 Two Pass Algorithm to Count $K_{3,3}$'s

We implemented a 2-pass algorithm to count $K_{3,3}$'s. Our implementation assumes the following input encoding of a directed input graph: The file contains n lists, one for each node v . The first lines of the list specify the indegree and outdegree of v . The following lines provide a list of edges pointing to v .

The first pass serves the purpose of counting the number of $K_{3,1}$'s in the graph. This way we avoid the evaluation of the algorithm for a logarithmic number of guesses. Our algorithm can easily be extended to a one-pass algorithm using the guessing technique described in detail in Section 5.

In the second pass we uniformly select s random $K_{3,1}$ subgraphs of the graph: We first sample a random index k between 1 and the number of $K_{3,1}$'s and retrieve the k -th $K_{3,1}$ from the stream using the algorithm UNIFORM $k_{3,1}$ (presented in Section 5).

After the selection of the $K_{3,1}$'s we select uniformly at random two vertices x_1, x_2 from a, b, c and two numbers k_1, k_2 in $\{1, 2, \dots, \Delta\}$. It remains to detect the vertices w and v , and to check the existence of 6 different edges per instance. This can both be done using the hashing technique described above to speed up the detection of vertices and edges which are interesting for one of s parallel instances.

7 Computational Experiments

This section presents the computational experiments performed by running the 1-pass algorithm to estimate the clustering coefficient of a graph and the 2-pass algorithm to count the number of $K_{3,3}$'s.

The codes were written in C/C++ and compiled with the `gcc` compiler version 3.2.2, using the `-O3` optimization option. The experiments were performed on a 2.4 GHz Intel Pentium IV computer with 512 MB of RAM, running Linux. Due to space requirements, the experiments for the webgraph were performed on a 2.8 GHz Intel Pentium IV computer with 1 GB of RAM, running Linux. The implementations are available by e-mail request.

CPU times were measured with the system function `getrusage`. The time for reading the graphs is not included in the reported running times.

7.1 Datasets

The datasets are divided into three subsets. The first set contains a large webgraph instance. It consists of 135 million nodes and 1 billion edges and is obtained from a graph extracted in 2001 by the WebBase project at Stanford [16]. We removed the frontier nodes, i.e, the nodes that have indegree equal to one and outdegree equal to zero.

The second set of instances contains the input graphs of the experiments of [17]. The instances are:

- `actor2004` and `actor2004`: Based on the *Internet Movie Database*. In these instances, two actors (nodes) are connected if they ever stared together in a movie.
- `authors`: Based on the *Computer Science Bibliography* at the University of Trier.
- `google-2002`: Based on the 2002 Google contest.
- `itdk0304`: The network of Internet routers (nodes) and their connections (edges) collected by the *Co-operative Association for Internet Data Analysis* (CAIDA).

The third set of instances represents the link structure of Wikipedia, as of an old dump of June 13, 2004 [3]. Wikipedia is nowadays the largest online encyclopedia, available in more than 200 languages. In these graphs, each article is a node and each hyperlink between nodes is a directed arc. A graph is extracted from each language. The experiments were performed on the graphs `wikiEN`, `wikiDE`, `wikiFR`, `wikiES`, `wikiIT` and `wikiPT`, extracted from the English, German, French, Spanish, Italian and Portuguese languages.

Table 1 presents basic properties of these graphs. All of them are sparse. The dimensions vary from less than ten thousand nodes to more than 135 million nodes, and from less than 100 thousand edges to over one billion edges.

7.2 Clustering Coefficient Computation

Table 2 presents results for the one pass algorithm to estimate the clustering coefficient of a graph. For directed graphs we replaced all directed edges by undirected edges and generated the complete adjacency list for each node. We ran the algorithm on each graph with two different sample sizes ($s = 300$ and $s = 1500$, and $s = 3000$).

For most of the graphs, the data stream algorithm was able to find an estimated clustering coefficient very close to the exact one represented by `cc*` in the table. Most of the time was spent in the operation of searching nodes in an adjacency list (there are two such operations in the algorithm).

On the webgraph we were not able to compute the exact clustering coefficient. Having the good approximations of all other instances in mind, we expect the approximation guarantee to be around 2%.

7.3 $K_{3,3}$ computation

Table 3 presents results for the two pass algorithm to estimate the number of $K_{3,3}$'s of a directed graph. As it is common in this kind of computation [12, 14], we removed nodes having an outdegree of more than 50. On average 15% of the nodes and 42% of the edges were removed.

Table 1: Basic properties of the graphs. The columns correspond to the number of nodes ($|V|$), number of arcs ($|E|$), minimum degree of a node (min), average degree of the nodes (avg), and maximum degree of a node (max).

Graph	graph dimensions		degree		
	$ V $	$ E $	min	avg	max
WebGraph	135,765,866	1,186,742,657	1	15.91	7,885,507
actor2004	667,609	27,581,275	1	82.63	4,605
google-2002	394,510	480,259	1	2.43	1,160
actor2002	382,219	15,038,083	1	78.69	3,96
authors	307,971	831,557	1	5.40	248
itdk0304	192,244	609,066	1	6.34	1,071
wikiEN	327,914	4,811,393	1	29.35	47,123
wikiDE	114,809	1,907,891	1	33.24	5,962
wikiFR	41,745	577,781	1	27.68	7,651
wikiES	25,684	246,316	1	19.18	2,973
wikiIT	12,758	134,342	1	21.06	1,793
wikiPT	8,071	42,083	1	10.43	2,317

Using a sample size of 1,000,000 almost all runs were able to find $K_{3,3}$'s in the sample set. The number of samples that detect a $K_{3,3}$ varies considerably among the instances. Considering a sample size of 1,000,000, hundreds and thousands of $K_{3,3}$'s were found in almost all instances. This indicates that there are strongly related communities in the respective graphs. Exceptions are some instances from the third set (some `wiki` graphs), and `google-2002`. We observe that in all instances for which a non-zero number of $K_{3,3}$'s are found in all the runs, the values reported with sample size 100,000 are very close to those reported for sample size 1,000,000. Running times are very short in all runs, even for a sample size of 1,000,000. The hashing tricks described in Section 6 seem to lead to a very small dependency of the runtime on the sample size s .

8 Acknowledgements

We thank Thomas Schank and Dorothea Wagner for five of the instances they have used in their paper [17]. We also thank A. Broder and S. Muthukrishnan for helpful discussions and Debora Donato for helping to organize the webgraph files.

References

- [1] Noga Alon, Yossi Matias, and Mario Szegedy, *The space complexity of approximating the frequency moments*.
- [2] L. Buriol, G. Frahling, S. Leonardi, A. Marchetti-Spaccamela, and C. Sohler, *Counting triangles in data streams*, In Proceedings of the ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 06, 2006.
- [3] L.S. Buriol, Carlos Castillo, D. Donato, S. Leonardi, and S. Millozzi, *Temporal evolution of the wikigraph*, Proceedings of Web Intelligence (Hong Kong), IEEE CS Press., December 2006.
- [4] Don Coppersmith and Shmuel Winograd, *Matrix multiplication via arithmetic progressions*, Journal of Symbolic Computation 3 (1990), no. 9, 251–280.
- [5] Jeffrey Dean and Sanjay Ghemawat, *Mapreduce: Simplified data processing on large clusters.*, Proceedings of the 6th Symposium on Operating System Design and Implementation, 2004.

Table 2: Results for estimating the cluster coefficient in digraphs, considering sample set sizes of 300, 1500, and 3000 for all graphs. For each instance and sample size, the table presents results for the estimated clustering coefficient \tilde{cc} and the corresponding running time measured in seconds (Time (s)). Moreover, we run an own implementation to calculate the exact clustering coefficient (cc*) on all graphs of feasible size.

Graph	s=300		s=1500		s=3000		cc*
	\tilde{cc}	Time (s)	\tilde{cc}	Time (s)	\tilde{cc}	Time (s)	
webgraph	0.31	11,380.18	-	-	-	-	-
actor2004	0.77	41.94	0.76	381.00	0.77	817.33	0.76
google-2002	0.03	1.56	0.05	9.05	0.05	25.32	0.05
actor2002	0.74	34.11	0.75	305.27	0.78	669.48	0.78
authors	0.58	2.94	0.52	21.92	0.58	47.44	0.60
itdk0304	0.11	1.73	0.15	13.60	0.15	37.94	0.16
wikiEN	0.21	16.32	0.21	16.32	0.26	276.17	0.27
wikiDE	0.21	16.54	0.20	101.59	0.22	234.82	0.24
wikiFR	0.29	7.80	0.28	77.31	0.28	163.88	0.29
wikiES	0.23	6.07	0.24	50.69	0.28	163.88	0.27
wikiIT	0.27	8.23	0.25	49.72	0.27	113.33	0.30
wikiPT	0.40	3.68	0.36	23.53	0.38	48.76	0.37

- [6] 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.
- [7] C. Sohler G. Frahling, P. Indyk, *Sampling in dynamic data streams and applications*, 21st Annual Symposium on Computational Geometry, ACM Press, 2005, pp. 142–149.
- [8] 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.
- [9] Sudipto Guha, Nick Koudas, and Kyuseok Shim, *Data-streams and histograms*, ACM Symposium on Theory of Computing, 2001, pp. 471–475.
- [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*, Proc. of the 8th WWW Conference, 1999, pp. 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] L. Laura, S. Leonardi, S. Millozzi, and J.F. Sybeyn, *Algorithms and experiments for the webgraph*, Proc. European Symposium on Algorithms (ESA), 2002.
- [15] S. Muthukrishnan, *Computing on data streams*, 2005.
- [16] The Standard WebBase Project, <http://www-diglib.stanford.edu/festbed/doc2/webbase/>.
- [17] Thomas Schank and Dorothea Wagner, *Finding, counting and listing all triangles in large graphs, an experimental study*, WEA, 2005, pp. 606–609.

Table 3: Results for estimating the number of $K_{3,3}$ in a digraph, considering sample set sizes of 100,000 and 1,000,000. For each run, the table presents the number of $K_{3,3}$'s estimated by the algorithm ($\widetilde{K}_{3,3}$) and the corresponding running time measured in seconds (Time (s)).

Graph	s=100,000		s=1,000,000	
	$\widetilde{K}_{3,3}$	Time (s)	$\widetilde{K}_{3,3}$	Time (s)
Webgraph	7,880,146,700,948,425	218.38	7,593,595,911,823,028	253.96
	7,880,146,700,948,425	219.26	6,017,566,571,633,343	254.48
	6,447,392,755,321,438	230.89	7,808,509,003,667,075	311.46
actor2004	5138977070	4.54	5253176561	8.75
	4838452096	4.67	5481575542	9.99
	5199082066	4.75	5346339303	9.20
google-2002	8859112	1.19	4724860	5.98
	0	1.13	4134252	6.04
	8859112	1.12	5315467	6.28
actor2002	138317076	1.39	188762127	8.74
	154589673	1.32	174116790	7.47
	162725972	1.35	180625828	7.18
authors	27783122	1.40	31638738	6.79
	35154154	1.26	31978940	6.79
	27783122	1.35	30334632	6.87
itdk0304	2550135420	1.30	2370936715	9.98
	2653519288	1.33	2284783492	10.71
	2067677368	1.58	2426074778	9.94
wikiEN	0	1.86	0	8.63
	0	1.90	1663312320	8.59
	0	1.83	831656160	8.68
wikiDE	0	1.34	408172513	8.21
	1530646924	1.17	612258770	8.14
	0	1.29	510215641	8.26
wikiFR	293980344	0.85	88194104	8.32
	293980344	0.86	146990172	8.31
	0	0.83	176388206	8.21
wikiES	163764410	0.75	223811360	7.50
	327528819	0.65	234728987	7.52
	191058478	0.71	259293648	7.45
wikiIT	62773047	0.68	43941133	7.38
	41848698	0.69	43941133	7.39
	20924349	0.67	35571394	7.47
wikiPT	1582891085	0.81	1559913634	9.02
	1442473328	0.79	1529277032	8.92
	1442473328	0.80	1589273710	9.02

- [18] Duncan J. Watts and Steven H. Strogatz, *Collective dynamics of small-world networks*, Nature **393** (1998), 440–442.
- [19] 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.

A Algorithms estimating their own accuracy

The description of the streaming algorithms given in the paper depends directly on the value they try to estimate (the clustering coefficient C_G resp. the number of bipartite cliques $K_{3,3}$). A return value of these algorithms therefore does not give us a guarantee on the accuracy (Note that the return value is only an approximation, but the approximation guarantees depend on the real value).

We overcome this problem in the following way: Let X be the value we are trying to estimate (C_G resp. $K_{3,3}$). We are given a streaming algorithm `ALGORITHM`, which outputs an indicator variable $\beta \in \{0, 1\}$ having $\mathbf{E}[\beta] = X$. We construct a new algorithm `ALGORITHMWITHGUARANTEE`. It has a number s and a desired error probability ψ as input parameters and outputs a pair $(\tilde{X}, \tilde{\epsilon})$. It runs at most s instances of `ALGORITHM` in parallel and gives the guarantee that \tilde{X} is a $(1 \pm \tilde{\epsilon})$ -approximation of X with probability $1 - \psi$. We show the pseudocode of `ALGORITHMWITHGUARANTEE` in Figure 1.

```

ALGORITHMWITHGUARANTEE (  $s \in \mathbb{N}, \psi \in (0, 1)$  )
  Set  $r \leftarrow \lceil 2 \cdot \log(3/\psi) \rceil$ .
  Set  $t \leftarrow \lceil s/(4 \log(3/\psi)) \rceil$ .
  Run  $r \cdot t$  instances  $I_{i,j}$  of ALGORITHM in parallel, one for each  $i \in \{1, \dots, r\}$  and  $j \in \{1, \dots, t\}$ 
  Let  $\tilde{X}_{i,j}$  be the value returned by instance  $I_{i,j}$ .
  Set  $\tilde{X}_i \leftarrow \frac{1}{t} \cdot \sum_{j=1}^t \tilde{X}_{i,j}$ 
  Set  $\tilde{X} \leftarrow \text{median}_i(\tilde{X}_i)$ .
  Set  $\tilde{\epsilon} \leftarrow \sqrt{\frac{528}{X} \cdot \frac{1}{s} \cdot \log \frac{3}{\psi}}$ .
  return  $(\tilde{X}, \tilde{\epsilon})$ .

```

Figure 1: The streaming algorithm `ALGORITHMWITHGUARANTEE`

Lemma A.1 Let $(\tilde{X}, \tilde{\epsilon})$ be the output of algorithm `ALGORITHMWITHGUARANTEE` and fix a value of $\epsilon \in \mathbb{R}^+$. With probability $1 - \psi$ the following statements are true:

- $(1 - \tilde{\epsilon}) \cdot X < \tilde{X} < (1 + \tilde{\epsilon}) \cdot X$
- If $s \geq \frac{1056}{\epsilon^2} \cdot \frac{1}{X} \cdot \log(3/\psi)$, then the algorithm outputs $(\tilde{X}, \tilde{\epsilon})$ having $\tilde{\epsilon} \leq \epsilon$.

Proof : `ALGORITHMWITHGUARANTEE` outputs $(\tilde{X}, \tilde{\epsilon})$ with $\tilde{\epsilon} = \sqrt{\frac{528}{X} \cdot \frac{1}{s} \cdot \log \frac{3}{\psi}}$. Therefore we have $\tilde{X} = \frac{528}{\tilde{\epsilon}^2} \cdot \frac{1}{s} \cdot \log \frac{3}{\psi}$. Because of the choice of t it follows

$$\tilde{X} \geq \frac{132}{\tilde{\epsilon}^2} \cdot \frac{1}{t} . \quad (1)$$

From Markov's inequality it follows that

$$\forall i \in \{1, \dots, r\} \Pr[\tilde{X}_i \geq 11 \cdot \mathbf{E}[\tilde{X}_i]] \leq \frac{1}{11} .$$

Since for all i we have $\mathbf{E}[\tilde{X}_i] = X$ it follows:

$$\forall_{i \in \{1, \dots, r\}} \Pr[\tilde{X}_i \geq 11 \cdot X] \leq \frac{1}{11} .$$

Because of $\tilde{X} = \text{median}_i(\tilde{X}_i)$ we can only have $\tilde{X} \geq 11 \cdot X$ if for at least $r/2$ values of i we have $\tilde{X}_i \geq 11 \cdot X$. For each single value of i the probability for that is smaller than $1/11$ as shown above. The probability to have at least $r/2$ values of i fulfilling that equation is therefore bounded by:

$$\begin{aligned} \Pr[\tilde{X} \geq 11 \cdot X] &\leq \binom{r}{r/2} \cdot \left(\frac{1}{11}\right)^{r/2} \\ &\leq \left(\frac{e \cdot r}{r/2}\right)^{r/2} \cdot \left(\frac{1}{11}\right)^{r/2} = \left(\frac{2e}{11}\right)^{r/2} \leq \left(\frac{1}{2}\right)^{r/2} \leq \psi/3 . \end{aligned}$$

From inequality (1) we conclude

$$\Pr\left[\frac{132}{\tilde{\epsilon}^2} \cdot \frac{1}{t} \geq 11 \cdot X\right] \leq \psi/3$$

and finally

$$\Pr\left[X \leq \frac{11}{\tilde{\epsilon}^2} \cdot \frac{1}{t}\right] \leq \psi/3 .$$

In the following we therefore condition on the event that $X > \frac{11}{\tilde{\epsilon}^2} \cdot \frac{1}{t}$ (which happens with probability at least $1 - \psi/3$). Since $\mathbf{E}[\tilde{X}_{i,j}] = X$ and $\tilde{X}_{i,j} \in \{0, 1\}$, we have

$$\mathbf{Var}[\tilde{X}_{i,j}] \leq \mathbf{E}[\tilde{X}_{i,j}^2] = \mathbf{E}[\tilde{X}_{i,j}] = X$$

and by the independence of the $\tilde{X}_{i,j}$:

$$\mathbf{Var}[\tilde{X}_i] = \mathbf{Var}\left[\frac{1}{t} \cdot \sum_{j=1}^t \tilde{X}_{i,j}\right] = \frac{1}{t^2} \cdot \sum_{j=1}^t \mathbf{Var}[\tilde{X}_{i,j}] = \frac{X}{t} . \quad (2)$$

By Chebyshev inequality:

$$\Pr[|\tilde{X}_i - X| \geq \tilde{\epsilon}X] \leq \frac{\mathbf{Var}[\tilde{X}_i]}{(\tilde{\epsilon} \cdot \mathbf{E}[\tilde{X}_i])^2} = \frac{X}{t \cdot \tilde{\epsilon} \cdot X^2} = \frac{1}{t \cdot \tilde{\epsilon} \cdot X} \leq \frac{1}{11}$$

since we conditioned on the event that $X > \frac{11}{\tilde{\epsilon}^2} \cdot \frac{1}{t}$. Therefore each fixed \tilde{X}_i is a value in $(1 \pm \tilde{\epsilon}) \cdot X$ with probability at least $1 - \frac{1}{11}$.

Since \tilde{X} is set to the median of all \tilde{X}_i , we can only have $\tilde{X} \notin (1 \pm \tilde{\epsilon}) \cdot X$ if for at least $r/2$ of the \tilde{X}_i we have $\tilde{X}_i \notin (1 \pm \tilde{\epsilon}) \cdot X$. The probability for that is bounded by:

$$\begin{aligned} \Pr[\tilde{X} \notin (1 \pm \tilde{\epsilon}) \cdot X] &\leq \binom{r}{r/2} \cdot \left(\frac{1}{11}\right)^{r/2} \\ &\leq \left(\frac{e \cdot r}{r/2}\right)^{r/2} \cdot \left(\frac{1}{11}\right)^{r/2} = \left(\frac{2e}{11}\right)^{r/2} \leq \left(\frac{1}{2}\right)^{r/2} \leq \psi/3 . \end{aligned}$$

The first statement of the lemma follows directly with probability $1 - 2 \cdot \frac{\psi}{3}$.

We now show the second statement. We condition on the event that $\tilde{X} \leq (1 + \tilde{\epsilon})X$, which happens with probability $1 - 2 \cdot \psi/3$ by the proofs above.

We assume the condition of the second statement that $s \geq \frac{1056}{\epsilon^2} \cdot \frac{1}{\bar{X}} \cdot \log \frac{3}{\psi}$. It follows directly $t \geq \frac{44}{\bar{X}}$. Therefore we get by Equation (2) and Chebyshev's inequality:

$$\Pr[|\tilde{X}_i - X| \geq \frac{1}{2} \cdot X] \leq \frac{\mathbf{Var}[\tilde{X}_i]}{\frac{1}{4} \cdot \mathbf{E}[\tilde{X}_i]^2} = \frac{4X}{t \cdot X^2} = \frac{4}{t \cdot X} \leq \frac{1}{11} .$$

By the median argument above we get

$$\Pr[\tilde{X} \leq \frac{X}{2}] \leq \Pr[|\tilde{X} - X| \geq \frac{1}{2} \cdot X] \leq \frac{\psi}{3} .$$

Together with $s \geq \frac{1056}{\epsilon^2} \cdot \frac{1}{\bar{X}} \cdot \log \frac{3}{\psi}$ we obtain with probability $1 - \psi$:

$$\tilde{X} \geq \frac{X}{2} \geq \frac{528}{\epsilon^2} \cdot \frac{1}{s} \cdot \log \frac{3}{\psi} .$$

It follows directly:

$$\epsilon \geq \sqrt{\frac{528}{\tilde{X}} \cdot \frac{1}{s} \cdot \log \frac{3}{\psi}} = \tilde{\epsilon} .$$

□

Notice that $r \cdot t \leq s$ for $s \geq 8 \log(3/\psi)$. Therefore we have at most s instances of ALGORITHM running in parallel. The time and space ALGORITHMWITHGUARANTEE(s) needs to process an edge in the stream is therefore at most s times the time and space of ALGORITHM.