Triangle counting in streamed graphs via small vertex covers

David García–Soriano* Konstantin Kutzkov†

Abstract

We present a new randomized algorithm for estimating the number of triangles in massive graphs revealed as a stream of edges in arbitrary order. It exploits the fact that graphs arising from various domains often have small vertex covers, which enables us to reduce the space usage and sample complexity of triangle counting algorithms. The algorithm runs in four passes over the edge set and uses constant processing time per edge.

We obtain precise bounds on the complexity and the approximation guarantee of our algorithm. For graphs where even the minimum vertex cover is prohibitively large, we extend the approach to achieve a trade-off between the number of passes and the space usage. Experiments on real-world graphs validate our theoretical analysis and show that the new algorithm yields more accurate estimates than state-of-the-art approaches using the same amount of space.

Keywords: *Stream mining, graph mining, triangle counting*

1 Introduction

Graphs provide abstract models to represent relationships between real-life objects. Certain properties of their structure can then give a better insight into the nature of the original problem. In the analysis of social networks, some widely used statistics are the number of triangles in a graph and the closely related transitivity and clustering coefficients of the graph. Applications include (but are not limited to) the discovery of hidden thematic structures in the World Wide Web [16], spam detection [4], motif detection in biological networks [36], Computer-Aided Design applications [17] and the analysis of the "small world phenomenon" [34].

Given the wide range of applications of triangle counting, it is not surprising that efficient computations of the number of triangles in a graph have become a major research topic. The naïve method solves the problem by considering every vertex triple and has thus complexity of $\Theta(n^3)$, where n is the number of vertices in the graph. The fastest algorithms for exact triangle counting are based on (fast) matrix multiplication, but such algorithms are mainly of theoretical importance due to the high constants involved and also to the fact that classic solutions assume that the graph fits in main memory. As a result of the huge amounts of data to be analyzed, approximate triangle counting in massive graphs has attracted a lot of attention in recent years. For large networks, the *streaming* model of computation that allows one to read the input sequentially and use only a small amount of memory has become ubiquitous. See Section 2.2 for a more detailed overview of known results and the techniques developed.

A classic topic in algorithmic graph theory is the minimum vertex cover problem. Given a graph $G = (V, E)$, the problem asks for the smallest possible set of vertices $\mathcal{C} \subseteq V$ such that each edge has at least one endpoint in $\mathcal{C}$. The problem is one of the 21 NP-hard problems presented by Karp [21] and has been intensively studied in terms of exact, approximation and parameterized algorithms. It is also known that certain classes of real-life graphs have small vertex covers [10]. In a recent work Green and Bader [14] showed that exact triangle counting can be considerably sped up for graphs with a small cover. However, to the best of our knowledge, the problem of designing efficient triangle counting algorithms for streamed graphs with small vertex covers has not yet been considered.

Organization of the paper. In Section 2 we give the necessary definitions, review previous work and discuss our contribution. In Section 3 we present the new algorithm. In 3.1 we start with an informal overview of the approach, in 3.2 we present the algorithm and in 3.3 we prove our main result. Motivated by observations on real graphs, in Section 4 we extend the algorithm to achieve a trade-off between the number of passes and the space complexity. In Section 5 we present an experimental evaluation of the algorithm. The paper is concluded in Section 6 where we also discuss some further research directions.

2 Preliminaries

2.1 Notation. Let $G = (V, E)$ be a simple undirected graph with n vertices and m edges. We will denote an edge $e = \{u, v\}$ connecting the vertices u and v by (u, v), or (v, u); we call u and v the *endpoints* of e. Vertex u is *neighbor* of v and vice versa, and $\mathcal{N}_u$ is the set of neighbors of u. The *degree* of u is $N_u = |\mathcal{N}_u|$.

*davidgs@yahoo-inc.com, Yahoo Labs, Barcelona, Spain.
†kutzkov@gmail.com, IT University of Copenhagen, Denmark.

A set of vertices $\mathcal{C} \subseteq V$ is called a *vertex cover* of G if $\forall e \in E : e \cap \mathcal{C} \neq \emptyset$. For a fixed cover $\mathcal{C}$ we write $\mathcal{U} = V \backslash \mathcal{C}$. Given a vertex u, we distinguish between the neighbours of u in $\mathcal{C}$ and $\mathcal{U}$: we set $\mathcal{N}_u^{\mathcal{C}} = \mathcal{N}_u \cap \mathcal{C}$ and $\mathcal{N}_u^{\mathcal{U}} = \mathcal{N}_u \cap \mathcal{U}$, and similarly $N_u^{\mathcal{C}} = |\mathcal{N}_u^{\mathcal{C}}|$, $N_u^{\mathcal{U}} = |\mathcal{N}_u^{\mathcal{U}}|$ for the number of such neighbors.

A *matching* in G is a set of edges $\mathcal{M} \subseteq E$ such that $\forall e_i, e_j \in \mathcal{M}, i \neq j : e_i \cap e_j = \emptyset$. The matching $\mathcal{M}$ is *maximal* if it is not contained in any other matching. A k-clique in G is a subset of k vertices $v_1, \ldots, v_k$ of G such that $(v_i, v_j) \in E$ for all $1 \leq i < j \leq k$. A 3-clique is called a *triangle*. A *2-path centered at* v is a tuple $(v, \{u, w\})$, where $u, v, w \in V$ and both (u, v) and (v, w) are in E. We write such a 2-path as (u, v, w); its *endpoints* are u and w. 2-paths will be also called *wedges*. We denote by $\mathcal{T}_3$ the set of triangles in G and their number is $T_3 = |\mathcal{T}_3|$. The set of 2-paths in G is $\mathcal{P}_2$ and $P_2 = |\mathcal{P}_2|$. The *transitivity coefficient* of G is

$$\tau(G) = \frac{3T_3}{\sum_{v \in V} \binom{N_v}{2}} = \frac{3T_3}{P_2},$$

i.e., the ratio of the number 2-paths in G contained in a triangle to the number of all 2-paths in G. When clear from context, we will omit G. For a given vertex cover $\mathcal{C}$ we extend the notation as follows: the set (the number) of triangles with two and three vertices in $\mathcal{C}$ is denoted $\mathcal{T}_3^{\mathcal{C},2}$ ($T_3^{\mathcal{C},2}$) and $\mathcal{T}_3^{\mathcal{C},3}$ ($T_3^{\mathcal{C},3}$), respectively. The set (number) of 2-paths with three vertices in $\mathcal{C}$ is denoted $\mathcal{P}_2^{\mathcal{C},3}$ ($P_2^{\mathcal{C},3}$). The set (number) of 2-paths with two vertices in $\mathcal{C}$ *and* centered at a vertex in $\mathcal{C}$ is $\mathcal{P}_2^{\mathcal{C},2}$ ($P_2^{\mathcal{C},2}$). The transitivity coefficient is then extended to $\tau_i = iT_3^{\mathcal{C},i}/P_2^{\mathcal{C},i}$, $i = 2, 3$.

An (ε, δ)-*approximation algorithm* $\mathcal{A}$ for some quantity q returns a value $\tilde{q}$ such that $(1-\varepsilon)q \leq \tilde{q} \leq (1+\varepsilon)q$ with probability at least $1 - \delta$. (Sometimes we say that $\mathcal{A}$ returns an (ε, δ)-approximation.)

Streaming model of computation. We assume the streaming model of computation as introduced in [15]. In this model the input stream $\mathcal{S}$ consists of N items $i_1, i_2, \ldots, i_N$. We are interested in computing some information about the input stream by sequentially scanning $\mathcal{S}$ only a constant number of times and using $o(N)$ machine words of random access memory. (For simplicity, the space complexity of the algorithm will be given as the number of machine words used instead of the more precise number of bits. For better readability of the results, we make the simplifying and certainly realistic assumption that each item can be described in constant number of machine words.) We also expect our algorithms to be practical, therefore we additionally require fast processing time per item when scanning $\mathcal{S}$.

Streamed graphs. The items in the input stream $\mathcal{S}$ are the graph edges. In the literature two models have been considered. In the *incidence list* stream model for each vertex $u \in V$ of degree d the edges $(u, v_1), \ldots, (u, v_d)$, i.e., all edges adjacent to u, are revealed in succession. In the *adjacency stream* model edges arrive in arbitrary order. Clearly, the incidence list model provides more information and algorithms analyzed in this model are usually better than algorithms assuming the adjacency stream model. Also, a distinction is made between algorithms performing a *single pass* over the input and algorithms which are allowed to read the input more than once and perform *several passes*. The latter requires that the input is either stored persistently on a secondary device or one can generate the stream several times and this constrains the range of applications.

2.2 Previous work. *Exact triangle counting.* The best known exact algorithm [1] uses fast matrix multiplication as a subroutine and runs in time $O(n^\omega)$, where ω is the matrix multiplication exponent, $\omega \leq 2.3727$ [32]. The algorithm also counts exactly the number of triangles per vertex. Note however that this algorithm is mainly of theoretical importance since it requires random access to the graph, and thus the graph has to fit in memory. Also, the currently known asymptotically fastest matrix multiplication algorithms do not admit an efficient implementation for realistic input sizes. For applications where the exact number of triangles is needed, theoretically slower triangle *listing* algorithms are used, see [11] for a discussion.

Approximate triangle counting in streamed graphs. It has been shown that exact triangle counting algorithms have to store $\Omega(m)$ edges in main memory in order to distinguish between graphs with 0 or 1 triangles [23]. However, for real graphs with many triangles a good approximation of their number often suffices. Several authors presented sampling or sketching based algorithms for the estimation of the number of triangles in a streaming setting [3, 4, 7, 12, 20, 22, 25, 28, 30, 31]. Building upon results from linear algebra, researchers proposed techniques for approximate triangle counting not relying on sampling [2, 29] but they appear not to be practical. The closely related problem of estimating the clustering coefficient was considered in [6, 24]. Motivated by the problem of detection of emerging web communities by analyzing the Web graph [23], Buriol et al. [6] studied the problem of counting bipartite cliques of small size. The more general problem of estimating the number of graph minors of fixed size was studied by Bordino et al [5].

The best triangle counting algorithm for adjacency

streams is due to Jha et al. [18]. It works by sampling edges at random from the graph stream. For a sufficient number of sampled edges, by the birthday paradox one then expects the presence of many 2-paths, or wedges, consisting of two edges in the sample. Using reservoir sampling, one maintains a random sample of edges and a random sample of 2-paths from the graph stream processed so far, and for each 2-path in the sample one checks whether it will be completed to a triangle by an incoming edge. This allows to compute an approximation of the transitivity coefficient. The number of 2-paths in the sample is used to estimate the total number of 2-paths in the graph, thus one obtains an estimate of the number of triangles. The approach works in only one pass over the input stream and thus has a wider range of applications.

2.3 Our contribution. Seshadhri et al. [28] showed that sampling of wedges uniformly at random yields excellent results for estimating the transitivity coefficient and other graph metrics. This can be explained by the fact that for most real graphs the transitivity coefficient is a large constant; see, e.g., [28]. Also, it has been observed [33, 35] that the transitivity coefficient remains relatively stable in evolving networks over time. Thus, given the lower bound on uniform sampling from [8], one can argue that for real graphs the space complexity of the wedge sampling approach, namely $O(\frac{1}{\tau\varepsilon^2}\log\frac{1}{\delta})$, is the optimal achievable by an algorithm that works by sampling triples of vertices.

The main contribution of the paper is to address the problem of triangle counting by wedge sampling in graphs provided as adjacency streams. For graphs with a small cover we design an algorithm whose space complexity is close, or even matches, the space complexity of the algorithm of the wedge sampling approach [28]. Graphs with small vertex cover are ubiquitous for many domains [10]. The algorithm runs in four passes over the edges. However, for some real-life graphs the vertex cover is prohibitively large, thus we extend the algorithm to achieve a trade-off between the number of passes and the space usage by allowing sequential writing to an external device. We experimentally show that when several passes over the edge set are allowed, our algorithm is a better alternative to known approaches using the same amount of space.

3 The new approach

3.1 The main idea. The classic algorithm for approximating the size of a minimum vertex cover, usually attributed to Gavril and Yannakakis, works as follows. We start with an empty matching of edges $\mathcal{M}$ and add a new edge $e \in E$ to $\mathcal{M}$ if it does not intersect any edge in $\mathcal{M}$. After all edges have been processed, the set $\mathcal{M}$ is a maximal matching. The vertices in $\mathcal{M}$ cover all edges in E, otherwise the matching would not be maximal. Also, the number of vertices in $\mathcal{M}$ is at most twice the cardinality of a minimum vertex. This follows from the fact that a minimum vertex cover must include at least one vertex from each edge in $\mathcal{M}$.

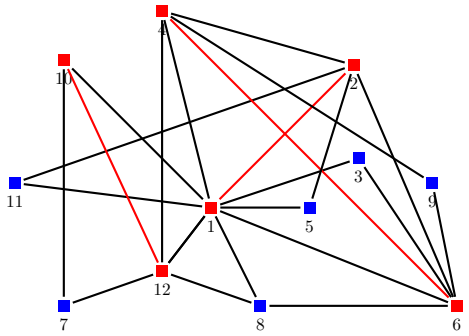

Figure 1: A toy example illustrating how the algorithm works. The red edges correspond to the maximal matching found by the algorithm and the red vertices are the vertex cover.

Our approach is based on the following observations. The minimum vertex cover 2-approximation algorithm works in a streaming fashion by sequentially reading the edges. The order in which the edges are provided is immaterial in order to achieve a 2-approximation. A vertex cover of G must cover at least two vertices in each triangle in G, otherwise there will exist an uncovered edge. So the basic idea is the following. In a single pass over E we compute a maximal matching of G, which yields a vertex cover $\mathcal{C}$ with cardinality at most twice the size of a minimum vertex cover. Then, in subsequent passes we use $\mathcal{C}$ to sample uniformly from the set of 2-paths with at least two vertices in $\mathcal{C}$ but without explicitly storing the neighborhoods of vertices in $\mathcal{C}$. Instead, for each $v \in \mathcal{C}$ we will only store the numbers $N_v^{\mathcal{C}}$ and $N_v^{\mathcal{U}}$, i.e., the number of neighbours of v in $\mathcal{C}$ and $\mathcal{U} = V \backslash \mathcal{C}$, respectively. We show that this will allow us to obtain an (ε, δ)-approximation of τ_2 and τ_3, respectively, and that we can compute the exact values of $P_2^{\mathcal{C},2}$ and $P_2^{\mathcal{C},3}$. This yields our main result.

3.2 The algorithm. Pseudocode description of the CoverCount algorithm is given in Figure 2. We assume edges are provided in arbitrary order and we are allowed several passes over them. The algorithm performs four passes over the input. We explain how the algorithm works using the toy example in Figure 1. In the first pass we find a vertex cover $\mathcal{C}$ by running the maximal matching algorithm. From now on we assume

a total order on the vertices in $\mathcal{C}$. Let in our example graph the edges arrive in the order (1,2), (1,5), (2,5), (2,11), (10, 12), (1,4), (1,8), (1,11), (2,4), (4,6), (6,9), (4,12), (1,6), (3,6), (7,12), (6,8), ... It is easy to see that the edges (1,2), (10,12) and (4, 6) form a maximal matching, thus we have $\mathcal{C} = (1, 2, 10, 12, 4, 6)$ with this order.

In the second pass for each vertex $v \in \mathcal{C}$ we initialize two counters $N_v^{\mathcal{C}}$ and $N_v^{\mathcal{U}}$ recording the current number of neighbours of v in $\mathcal{C}$ and $\mathcal{U} = V \backslash \mathcal{C}$, respectively. Then we count the exact number of such neighbors; this is done by the procedure UPDATE COUNTERS which updates the respective counters for each incoming edge (u, v) (observe that at least one of u and v is in $\mathcal{C}$). In our example, after processing all edges we have $N_1^{\mathcal{C}} = 5$ and $N_1^{\mathcal{U}} = 4$, i.e., the vertex 1 has 5 neighbors in $\mathcal{C}$ and 4 neighbors in $\mathcal{U}$, while $N_6^{\mathcal{C}} = 3$ and $N_6^{\mathcal{U}} = 3$.

We next sample uniformly at random K 2-paths from $\mathcal{P}_2^{\mathcal{C},2}$ and $\mathcal{P}_2^{\mathcal{C},3}$. This is done by the procedure SAMPLE 2-PATHS. From the number of neighbors of each vertex in $\mathcal{C}$ we first compute the numbers $P_2^{\mathcal{C},2}$ and $P_3^{\mathcal{C},3}$. Clearly $P_2^{\mathcal{C},2} = \sum_{v \in \mathcal{C}} N_v^{\mathcal{C}} N_v^{\mathcal{U}}$ and $P_2^{\mathcal{C},3} = \sum_{v \in \mathcal{C}} \binom{N_v^{\mathcal{C}}}{2}$. Assume that the 2-paths in $\mathcal{P}_2^{\mathcal{C},2}$ and $\mathcal{P}_2^{\mathcal{C},3}$ are numbered as $0, 1, \ldots, P_2^{\mathcal{C},2} - 1$ and $0, 1, \ldots, P_2^{\mathcal{C},3} - 1$, respectively. Then we select uniformly at random and independently of each other K numbers from $\mathcal{P}_2^{\mathcal{C},2}$ and $\mathcal{P}_2^{\mathcal{C},3}$. For each such number r we determine the corresponding *2-path number*. Thus is done by first determining the vertex $v \in \mathcal{C}$ at which the 2-path is centered. Assuming total order on the vertices in $\mathcal{C}$, the r-th 2-path in $\mathcal{P}_2^{\mathcal{C},2}$ is centered at the k-th vertex for which it holds that

$$\sum_{u=0}^{k} N_u^{\mathcal{C}} N_u^{\mathcal{U}} \leq r < \sum_{u=k+1}^{|\mathcal{C}|} N_u^{\mathcal{C}} N_u^{\mathcal{U}}$$

and the r-th 2-path in $\mathcal{P}_2^{\mathcal{C},3}$ is centered at the k-th vertex in $\mathcal{C}$ for which it holds that

$$\sum_{u=0}^{k} \binom{N_u^{\mathcal{C}}}{2} \leq r < \sum_{u=k+1}^{|\mathcal{C}|} \binom{N_u^{\mathcal{C}}}{2}.$$

Assume that the r-th 2-path is centered at v. Let the 2-paths centered at v be numbered according to the numbers of the edges incident to v, and the edges be numbered according to the order of appearance in the stream as $0, 1, \ldots, N_v - 1$: the 2-path numbered as 0 with three vertices in $\mathcal{C}$ is thus written as $(\mathcal{N}_v^{\mathcal{C}}[0], v, \mathcal{N}_v^{\mathcal{C}}[1])$ denoting that it consists of the 0th and first edge incident to v with endpoints in $\mathcal{C}$. Similarly, the 0th 2-path centered at v with one endpoint in $\mathcal{C}$ and one endpoint in $\mathcal{U}$ is implicitly given by $(\mathcal{N}_v^{\mathcal{C}}[0], v, \mathcal{N}_v^{\mathcal{U}}[0])$. With some simple algebra one then obtains the formulas for the number of neighbors which determine the r-th 2-path in DETERMINE 2-PATH. Note that the procedure does not require a new pass over the edges and we still don't know the vertices in the sampled 2-paths, but rather an implicit description like "the 2-path centered at vertex v with endpoints the second and seventh vertices in $\mathcal{N}_u^{\mathcal{C}}$." These "implicit" 2-paths are stored in hash tables $\mathcal{R}_2$ and $\mathcal{R}_3$, respectively. We explain later the exact way of storing them.

Let us consider again our toy example. It holds $P_2^{\mathcal{C},2} = 47$. Assume we want to determine the 2-path numbered 30 in $\mathcal{P}_2^{\mathcal{C},2}$. We remind that the vertices in $\mathcal{C}$ are ordered as 1, 2, 10, 12, 4, 6. There are 20 such 2-paths centered at vertex 1, 6 2-paths centered at 2 and 2 2-paths centered at vertex 10 with exactly two vertices from $\mathcal{C}$. Since there are 6 2-paths centered at 12, we conclude that the 2-path we are looking for is centered at vertex 12 and among those 2-paths we are looking for the 2-path with the number $30 - 28 = 2$. Thus we conclude that the implicit 2-path is "($\mathcal{N}_{12}^{\mathcal{C}}[0]$, 12, $\mathcal{N}_{12}^{\mathcal{U}}[1]$)".

Assume the sampled K 2-paths in $\mathcal{R}_2$ and $\mathcal{R}_3$ are numbered from 0 to $K-1$. Then we store in hash tables $\mathcal{R}_2$ and $\mathcal{R}_3$ the pairs $((u, \mathcal{N}_u^{\mathcal{C}}[i]), k)$, $0 \leq k \leq K-1$, denoting that the edge between u and the i-th neighbor in $\mathcal{N}_u^{\mathcal{C}}$ is part of the k-th sample. In the third pass we count for each $u \in \mathcal{C}$ the number of its neighbors in $\mathcal{C}$ and $\mathcal{U}$ and check in $\mathcal{R}_2$ and $\mathcal{R}_3$ whether the last edge seen is part of a sampled 2-path. We update the implicit representation of a sampled 2-path such that it records the vertices it contains. In our example we look for the "0th vertex in $\mathcal{N}_{12}^{\mathcal{C}}$".Given the order in which edges arrive we thus find that the 30th 2-path in $\mathcal{P}_2^{\mathcal{C},2}$ is $(10, 12, 7)$.

After all $2K$ sampled 2-paths have been constructed, we build a hash table $Missing$ containing entries $((u, v), s_2, s_3)$, $s \geq 1$. The meaning of this entry is that if $(u, v) \in E$, then s_2 sampled 2-paths can be completed to a triangle in $\mathcal{T}_3^{\mathcal{C},2}$ and s_3 samples to a triangle in $\mathcal{T}_3^{\mathcal{C},3}$. In a final pass we check which of the sampled 2-paths are part of a triangle and update the corresponding counters.

In our example we look for the edge $(10, 7)$. In $Missing$ we have the entry $((10, 7), 1, 0)$, denoting that the existence of the edge $(10, 7)$ implies one triangle in $\mathcal{T}_3^{\mathcal{C},2}$. We see that $(10, 7) \in E$ and update the corresponding counter.

3.3 Analysis.

THEOREM 1. *Let $G = (V, E)$ be a graph with m edges and n vertices. Let C be the cardinality of a minimum*

1st Pass

Input: stream $\mathcal{S}$ edges in arbitrary order
1: $\mathcal{C} = \emptyset$
2: **for** each (u, v) in $\mathcal{S}$ **do**
3: **if** $\mathcal{C} \cap \{u, v\} = \emptyset$ **then**
4: $\mathcal{C} = \mathcal{C} \cup \{u, v\}$
5: **return** $\mathcal{C}$

2nd Pass

Input: stream $\mathcal{S}$ edges, vertex cover $\mathcal{C}$
1: Initialize counters($\mathcal{C}$)
2: **for** each (u, v) in $\mathcal{S}$ **do**
3: Update counters($\mathcal{C}, (u, v)$)

3rd Pass

Input: stream $\mathcal{S}$ edges, vertex cover $\mathcal{C}$, parameter K
1: $\mathcal{R}_2$ = Sample 2-paths($\mathcal{C}, K, 2$)
2: $\mathcal{R}_3$ = Sample 2-paths($\mathcal{C}, K, 3$)
3: Initialize counters($\mathcal{C}$)
4: **for** each (u, v) in $\mathcal{S}$ **do**
5: Update counters($\mathcal{C}, (u, v)$)
6: **if** $u \in \mathcal{C}$ **then**
7: Update samples($u, v, \mathcal{R}_2, \mathcal{R}_3$)
8: **if** $v \in \mathcal{C}$ **then**
9: Update samples($v, u, \mathcal{R}_2, \mathcal{R}_3$)

4th Pass

Input: stream $\mathcal{S}$ edges in arbitrary order, samples of 2-paths $\mathcal{R}_2$ and $\mathcal{R}_3$
1: *Missing* =Get Missing edges($\mathcal{R}_2, \mathcal{R}_3$)
2: $\ell_2 = 0$, $\ell_3 = 0$
3: **for** each (u, v) in $\mathcal{S}$ **do**
4: **if** $((u, v), s_2, s_3) \in$ *Missing* **then**
5: $\ell_2 = \ell_2 + s_2$
6: $\ell_3 = \ell_3 + s_3$
7: **return** $\frac{\ell_2 P_2^{\mathcal{C},2}}{2K} + \frac{\ell_3 P_2^{\mathcal{C},3}}{3K}$

Figure 2: Pseudocode description of the four passes of CoverCount.

vertex cover of G, T_3 the number of triangles in G and τ the transitivity coefficient of G. There exists an algorithm computing an (ε, δ)-approximation of T_3 in four passes over E which uses space $O(C + \frac{1}{\varepsilon^2 \tau} \log \frac{1}{\delta})$ and runs in time $O(m + \frac{1}{\varepsilon^2 \tau} \log \frac{1}{\delta})$.

Proof. We first analyze the complexity of the algorithm. As discussed, we have $|\mathcal{C}| \leq 2C$. We implement $\mathcal{C}$ as a hash table with (amortized) constant update and look-up time, thus the first two passes run in $O(m)$ time and $O(C)$ space. We assume access to a source of fully random numbers, thus the $2K$ numbers of sampled 2-paths are generated in time $O(K)$. We find the implicit representation of the sampled 2-paths by first sorting the K numbers and then sequentially reading the list of vertices in $\mathcal{C}$. (The K numbers can be sorted in time $O(K)$ by bucket sort.) In the third pass we count for each $u \in \mathcal{C}$ the number of its neighbors and check in $\mathcal{R}_2$ and $\mathcal{R}_3$ whether the last edge seen is part of a sampled

Initialize counters

Input: set of vertices $\mathcal{C}$
1: **for** each v in $\mathcal{C}$ **do**
2: $N_v^{\mathcal{C}} = 0$, $N_v^{\mathcal{U}} = 0$

Update counters

Input: vertex cover $\mathcal{C}$, edge (u, v)
1: **if** $u \in \mathcal{C}$ **then**
2: **if** $v \in \mathcal{C}$ **then**
3: $N_u^{\mathcal{C}}$++, $N_v^{\mathcal{C}}$++
4: **else**
5: $N_u^{\mathcal{U}}$++
6: **else**
7: $N_v^{\mathcal{U}}$++

Update samples

Input: vertex $u \in \mathcal{C}$, neighbor $v \in V$, vertex cover $\mathcal{C}$, samples $\mathcal{R}_2, \mathcal{R}_3$)
1: **if** $v \in \mathcal{C}$ **then**
2: **if** $\mathcal{R}_3$ contains $(u, N_u^{\mathcal{C}})$ or $\mathcal{R}_2$ contains $(u, N_u^{\mathcal{C}})$ **then**
3: Update $(u, N_u^{\mathcal{C}})$ to (u, v) in $\mathcal{R}_3$ or $\mathcal{R}_2$, respectively
4: **else**
5: **if** $\mathcal{R}_2$ contains $(u, n_u^{\mathcal{U}})$ **then**
6: Update $(u, N_u^{\mathcal{U}})$ to (u, v) in $\mathcal{R}_2$

Sample 2-paths

Input: set of vertices $\mathcal{C}$ together with the number of their neighbors in $\mathcal{C}$ and $V \backslash \mathcal{C}$, K–number of samples, nr–number of neighbors in $\mathcal{C}$
1: **if** $nr = 3$ **then**
2: $P_2^{\mathcal{C},3} = \sum_{v \in \mathcal{C}} \binom{N_v^{\mathcal{C}}}{2}$
3: Choose uniformly at random a set $\mathcal{R}$ of K numbers $r_1, \ldots, r_K$ from $\{0, 1, \ldots, P_2^{\mathcal{C},3} - 1\}$
4: **if** $nr = 2$ **then**
5: $P_2^{\mathcal{C},2} = \sum_{v \in \mathcal{C}} N_v^{\mathcal{C}} N_v^{\mathcal{U}}$
6: Choose uniformly at random K $r_1, \ldots, r_K$ numbers from $\{0, 1, \ldots, P_2^{\mathcal{C},2} - 1\}$
7: **for** each $r \in \mathcal{R}$ **do**
8: Determine 2-path($\mathcal{C}, r, n$) and update it in $\mathcal{R}$
9: **return** $\mathcal{R}$

Determine 2-path

Input: $\mathcal{C}$– an ordered list of vertices $\mathcal{C}$ together with the number of their neighbors, r – number of 2-path, nr–number of neighbors in $\mathcal{C}$
1: **if** $nr = 2$ **then**
2: Find $v \in \mathcal{C}$ s. t. $\sum_{u=0}^{v} N_u^{\mathcal{C}} n_u^{\mathcal{U}} \leq k < \sum_{u=0}^{v+1} N_u^{\mathcal{C}} N_u^{\mathcal{U}}$
3: $k = k - \sum_{u=0}^{v} N_u^{\mathcal{C}} N_u^{\mathcal{U}}$
4: Let $d = N_{v+1}^{\mathcal{U}}$
5: $i = \lfloor k/d \rfloor$, $j = k - id$
6: **if** $nr = 3$ **then**
7: Find $v \in \mathcal{C}$ s. t. $\sum_{u=0}^{v} \binom{N_u^{\mathcal{C}}}{2} \leq k < \sum_{u=0}^{v+1} \binom{N_u^{\mathcal{C}}}{2}$
8: $k = k - \sum_{u=0}^{v} \binom{N_u^{\mathcal{C}}}{2}$
9: Let $d = N_{v+1}^{\mathcal{C}}$
10: Let x^* be the minimum x such that $x^2 - x - d^2 + d + 2k \geq 0$.
11: $i = d - x^*$, $j = k - id + i(i+1)/2 + i + 1$
12: **return** (i, v, j)

Get missing edges

Input: 2-path samples $\mathcal{R}_2$, $\mathcal{R}_3$
1: **for** each 2-path $(u, v, w) \in \mathcal{R}_2$ **do**
2: *Missing.update*$((u, w), s_2 + 1, s_3)$
3: **for** each 2-path $(u, v, w) \in \mathcal{R}_3$ **do**
4: *Missing.update*$((u, w), s_2, s_3 + 1)$
5: **return** *Missing*

2-path, thus each edge is processed in constant time. Building the hash table $Missing$ needs time and space $O(K)$. For each edge, checking whether it is in $Missing$ takes constant time. The running time of the last two passes is thus $O(m+K)$ and space complexity is $O(K)$.

The correctness of the algorithm is straightforward. The numbers $P_2^{\mathcal{C},2}$ and $P_2^{\mathcal{C},3}$ can be computed exactly, thus we analyze for what value of K we will obtain an (ε,δ)-approximation of τ_2 and τ_3. The probability that a 2-path chosen at random from all 2-paths with three vertices in $\mathcal{C}$ is part of a triangle is $\tau_3 = 3T_3^{\mathcal{C},3}/P_2^{\mathcal{C},3}$. Similarly, the probability that we choose at random a 2-path with two vertices in $\mathcal{C}$ which is part of a triangle is $\tau_2 = 2T_3^{\mathcal{C},2}/P_2^{\mathcal{C},2}$. By Chernoff's inequality we can take

$$K = \max\Big(O(\frac{1}{\tau_2\varepsilon^2}\log\frac{1}{\delta}), O(\frac{1}{\tau_3\varepsilon^2}\log\frac{1}{\delta})\Big)$$

in order to obtain an (ε,δ)-approximation of $\tau^{\mathcal{C},3}$ and $\tau^{\mathcal{C},3}$, respectively. We have

$$\tau_2+\tau_3 \geq \frac{2T_3}{P_2} = 2\tau/3.$$

Assume w.l.o.g. that $\tau_2 = 2\alpha\tau/3$ and $\tau_3 \geq 2(1-\alpha)\tau/3$ for some $\alpha \leq 1/2$. Therefore, in order to obtain an $(1\pm\varepsilon)$-approximation of τ with probability more than $1-\delta$, we need an $(1\pm\varepsilon/(2\alpha))$-approximation of τ_2 and an $(1\pm\varepsilon/2)$-approximation of τ_3, each with probability at least $1-\delta/2$. By Chernoff's inequality for the first we can take

$$O\Big(\frac{\alpha^2}{\alpha\tau\varepsilon^2}\log\frac{1}{\delta}\Big) = O\Big(\frac{\alpha}{\tau\varepsilon^2}\log\frac{1}{\delta}\Big)$$

samples, and for the latter $O(\frac{1}{\tau\varepsilon^2}\log\frac{1}{\delta})$, thus we set

$$K = O\Big(\frac{1}{\tau\varepsilon^2}\log\frac{1}{\delta}\Big).$$

By the union bound this implies an $1\pm\varepsilon/2$-approximation of τ_2 and τ_3 with probability $1-\delta$. □

4 Multi-pass version of the algorithm

For the case when the dominating term of the space complexity of the algorithm is the storage requirement to keep the vertex cover in main memory rather than the $O(\frac{1}{\tau\varepsilon^2}\log\frac{1}{\delta})$ samples, a common situation for massive graphs and relatively small amount of main memory, we extend the algorithm to achieve better space complexity by increasing the number of passes and writing to a secondary device. The extension works in the *W-stream model* [27] where we are allowed to sequentially write to the input stream and the output stream of the i-th pass is then the input stream for the $(i+1)$-th pass. Graph stream algorithms in this model have been shown to be highly practical, see e.g. [4, 13].

THEOREM 2. *Let $G=(V,E)$ be a graph with m edges and n vertices. Let C be the cardinality of a minimum vertex cover of G, T_3 be the number of triangles in G and τ the transitivity coefficient of G. For any $t \geq 2$, there exists an algorithm computing an (ε,δ)-approximation of T_3 in $2t+3$ passes over E which uses $O(C/t+\frac{1}{\varepsilon^2\tau}\log\frac{1}{\delta})$ main memory words and running in time $O(mt+\frac{1}{\varepsilon^2\tau}\log\frac{1}{\delta})$.*

Proof. We modify the algorithm as follows. Assume we can store $2q$ words in memory. In the first pass we add vertices to a cover $\mathcal{C}_1$ until $|\mathcal{C}_1| \leq q$. Assume w.l.o.g. that the edge $(u,v) \in \mathcal{S}$ is covered by v, i.e., $v \in \mathcal{C}_1$. In a second pass we check for each edge $(u,v) \in \mathcal{S}$ which of u and v, if any, are covered by $\mathcal{C}_1$. Assume w.l.o.g. that $v \in \mathcal{C}_1$, then we overwrite the edge to $(u,v^{\mathcal{C}})$. ($v^{\mathcal{C}}$ denotes that the vertex v is part of the cover $\mathcal{C}$.) Simultaneously, we compute a matching $\mathcal{C}_2$ of cardinality at most q by adding edges not covered by $\mathcal{C}_1$. After processing in this way all edges in the stream in two passes, we have found a partial cover $\mathcal{C}_1 \cup \mathcal{C}_2$ of G and all edges covered by $\mathcal{C}_1$ are labeled correspondingly in the graph stream. In the next pass we repeat the above by computing a new partial cover $\mathcal{C}_3$ but only consider unlabeled edges, i.e., edges not covered by $\mathcal{C}_1 \cup \mathcal{C}_2$. Note that we don't need any longer to keep $\mathcal{C}_1$ in memory, but we need $\mathcal{C}_2$ such that we can label the edges covered by $\mathcal{C}_2$. In general, in the ith pass, $i \geq 2$, we compute a partial cover $\mathcal{C}_i$ of the edges of G not covered by $\mathcal{C}_1 \cup \ldots \cup \mathcal{C}_{i-1}$ and we label the edges in G covered by $\mathcal{C}_{i-1}$. Thus, after $t = \lceil \frac{2|\mathcal{C}|}{q} \rceil$ passes we have computed a vertex cover $\mathcal{C} = \mathcal{C}_1 \cup \ldots \cup \mathcal{C}_t$. In a final pass we label the edges covered by $\mathcal{C}_t$. Thus, after $t+1$ passes we have a graph stream with all vertices labeled by a vertex cover $\mathcal{C}$ whose size is a 2-approximation of the size of a minimum vertex cover.

Then we simulate the second pass of COVERCOUNT using t passes and main memory space bounded by $2|\mathcal{C}|/t$ with high probability. We want to divide the set of vertices in $\mathcal{C}$ in $q = \mathcal{C}/t$ disjoint subsets of equal cardinality (up to ± 1). For each such subset we will compute the number of neighbors of each vertex in $\mathcal{C}$ and write them to the input stream for the next pass. We define a hash function $h : \mathcal{C} \to (0,1]$. Then, in the ith pass, $1 \leq i \leq q$, we count the exact number of neighbors in $\mathcal{C}$ and $V \backslash \mathcal{C}$ of vertices in $\mathcal{C} \cap h^{-1}((i-1)/q, i/q]$, i.e, vertices in the cover with hash value in the interval $((i-1)/q, i/q]$. We implement h using tabulation hashing [9]: We view each key $v \in \mathcal{C}$ as a vector consisting of c characters, $v = (v_1, v_2, \ldots, v_c)$, where the ith character is from a universe $\mathcal{C}_i$ of size $|\mathcal{C}|^{1/c}$. (W.l.o.g. we assume that $n^{1/c}$ is an integer.) For each universe $\mathcal{C}_i$ we initialize a table T_i and for each

character $v_i \in V_i$ we store a random value $r_{v_i} \in [s]$. Then the hash value is computed as

$$h_0(v) = T_1[v_1] \oplus T_2[v_2] \oplus \ldots \oplus T_c[v_c]$$

where $\oplus$ denotes the bit-wise XOR operation. Thus, for a small constant c, the space needed is $O(n^{1/c} \log n)$ bits and the evaluation time is $O(1)$ array accesses. In what follows we assume $c > 2$. As shown in [26], with probability $1 - n^{-\gamma}$, for any constant $\gamma > 1$, in no interval will there be more than $q \pm O(\sqrt{q} \log^c |\mathcal{C}|)$ vertices. Since we assume t is constant, and thus $q = \omega(\log^c |\mathcal{C}|)$ for any fixed c, we assume that for a large vertex cover $\mathcal{C}$, in no interval will there be more than $2t$ vertices. Clearly, we can force the algorithm to store in memory at most q vertices in each pass: if there are more than q vertices with a hash value in a given interval $((i-1)/q, i/q]$, then we will consider only vertices in the interval $((i-1)/q, (i-1/2)/q]$ and in a subsequent pass vertices in the interval $((i-1/2)/q, i/q]$. Thus, we have worst case space usage of $2q$ and with high probability the number of passes is t.

Summing up, we can simulate the first two passes of CoverCount with $2t+1$ passes. In each pass we need at most $2|\mathcal{C}|/t$ words of main memory and the running time in each pass is $O(m)$. For the next pass we assume the stream consists of a labeled graph and a vertex cover C such that for each vertex $v \in \mathcal{C}$ we also have $N_v^{\mathcal{C}}$ and $N_v^{\mathcal{U}}$. We can then compute exactly the numbers $P_2^{\mathcal{C},2}$ and $P_2^{\mathcal{C},3}$, and sample $2K$ random 2-path numbers. By first sorting the 2-path numbers and then sequentially reading the vertices and the number of their neighbors in $\mathcal{C}$ and $V \backslash \mathcal{C}$, we compute an implicit representation of the sampled 2-paths. The last two passes are identical to CoverCount. □

5 Experiments

Datasets and implementation. The algorithm was implemented in Java and experiments performed on a Windows 7 machine with 2.20 GHz CPU and 4 GB RAM. For the generation of random 2-path numbers we used the random numbers from the Marsaglia Random Number CDROM[1].

In Table 5 we summarize the statistics about several real-life graphs from the SNAP library [2]. The graphs come from different domains. Oregon presents peering information inferred from Oregon route-views, AS-Caida – autonomous systems graphs, Email-EU is the network of emails sent between employees in an EU institution, Epinions is the graph of a who-trust-whom online social network, Web-NotreDame is a web graph from the University of Notre Dame pages, Wiki-Talk is network of users' edit pages and Youtube is the friendship graph among users of the video-sharing website. The vc column contains the size of the cover computed by the greedy maximal matching algorithm. As one can see, even the 2-approximation of the minimum vertex cover is often considerably smaller than the number of vertices.

We ran experiments on several graphs but choose to present detailed results about two graphs, Web-NotreDame and WikiTalk, which best illustrate the advantages and limitations of our approach.

Comparison with other approaches. It appears that there is no clear competitor to our algorithm. Other algorithms either assume the easier model where the graph is provided as incidence lists stream or the more restrictive model where only one pass over the graph stream is allowed. However, for many applications one can assume that the input graph is either persistently stored on a secondary device or the stream can be generated more than once by a deterministic procedure.

Therefore, we chose to compare CoverCount to Doulion [30], which is known to yield excellent results on real graphs, and to the recent algorithm by Jha et al. [18] which is the state-of-the-art triangle counting approach for adjacency streams. Doulion is a sparsification approach and works by sampling each edge with probability p and building a sparsified graph G_S. After processing all edges, the number of triangles in G is estimated as the $T_3(G_S)/p^3$. We implemented the modification of Doulion presented in [25]. Instead of sampling at random each incoming edge with probability p, we used a random coloring function $f : V \to C$ for a set of $\lfloor 1/p \rfloor$ colors C and keep an edge in the sparsified graph (u, v) iff $f(u) = f(v)$. Confirming the analysis from [25], this modification yields somewhat better results.

The algorithm by Jha et al. maintains a uniform sample of the edges processed so far using reservoir sampling. For a sufficiently large number of sampled edges s_e, one then shows that with high probability one can build $W = W(s_e)$ 2-paths (or wedges in the notation of [18]). From these W wedges one maintains a random sample of s_w wedges which are used to estimate the transitivity coefficient τ. Once we have computed an estimate $\tilde{\tau}$, the number of triangles is estimated as $\frac{w\tilde{\tau}m^2}{3s_e(s_e-1)}$, where w is the number of wedges that can be constructed from the sampled edges. Note that this means that we estimate *both* the transitivity coefficient and the total number of wedges in the graph stream.

We implemented the three algorithms such that they have roughly the same space complexity. (Note

[1] http://www.stat.fsu.edu/pub/diehard/cdrom/
[2] http://snap.stanford.edu/data/

Graph	m	n	$\tau(G)$	vc	T_3
Oregon	23,409	11,174	0.0096	2,974	19,894
AS-Caida	53,381	26,475	0.0073	6,648	36,365
Email-EU	364,481	265,214	0.0041	36,204	267,313
Epinions	405,740	75,879	0.0656	33,692	1,624,481
Web-NotreDame	1,090,108	325,729	0.0877	117,078	8,910,005
Wiki-Talk	4,659,565	2,394,385	0.0033	92,450	9,203,519
Youtube	2,987,624	1,134,890	0.0062	456,406	3,056,386

Table 1: Information on datasets.

that we don't give precise results for the space complexity of the three algorithms in terms of the number of bits used. This highly depends on low-level technical details that are beyond the scope of the paper.) For CoverCount we implemented the multi-pass version and allowed to store in each pass a partial cover consisting of 40,000 vertices in main memory. (This means that one needs $2\lceil vc/40,000\rceil+1$ passes over the graph in order to compute the vertex cover.) After the graph has been labeled, we sampled 40,000 random wedges from $\mathcal{P}_2^{\mathcal{C},2}$ and $\mathcal{P}_2^{\mathcal{C},3}$. We ran the Jha et al. algorithm with 40,000 sampled edges and maintained from them a sample of 40,000 wedges. For Doulion we chose a sampling probability p such that the expected number of sampled edges is above 40,000.

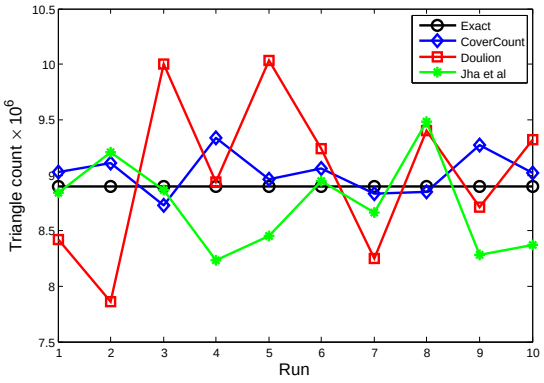

Figure 3: Estimates for the Web-NotreDame graph for 10 runs of the three algorithms.

Approximation guarantees. CoverCount is essentially an improvement of random wedge sampling, which yields unbiased estimation of the transitivity coefficient, adapted to streamed graphs with small vertex covers. We compute exactly the numbers $P_2^{\mathcal{C},2}$ and $P_2^{\mathcal{C},3}$, we thus expect high quality estimates. The plots in Figures 3 and 4 confirm this. We see that the algorithm of Jha et al. is pretty accurate for Web-NotreDame but sometimes considerably over- or underestimates the number of triangles in the WikiTalk graph. The reason is that Web-NotreDame has much higher transitivity coefficient than WikiTalk and even a small bias in the estimate of WikiTalk's transitivity coefficient can greatly affect the estimated number of triangles. Not surprisingly, for such a small sparsification ratios (0.025 for Web-NotreDame and 0.01 for WikiTalk) the errors of Doulion's estimates can be considerable.

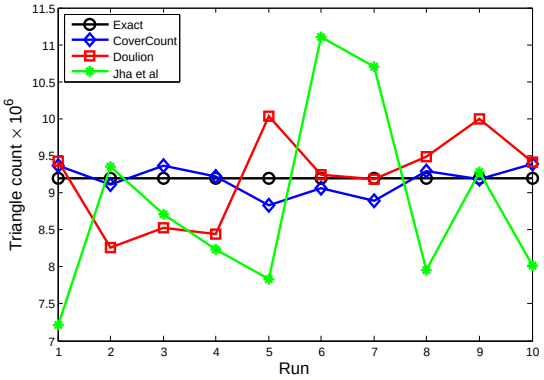

Figure 4: Estimates for the WikiTalk graph for 10 runs of the three algorithms.

Running time. The most time consuming phase is to label the graph with the vertex cover. For both Web-NotreDame and WikiTalk, CoverCount needs only seven passes over the graph in order to label the graph with the cover computed by a maximal matching, for a total number of nine passes over the input.

For only about 40,000 sampled edges Doulion is very efficient: the running time for the Web-NotreDame graph is just about a second and for WikiTalk about 4 seconds. CoverCount needed 18 seconds on Web-NotreDame and about a minute on WikiTalk. (The 4-pass version of CoverCount where we are allowed to keep the whole cover in memory is more efficient with 7 seconds for Web-NotreDame and 30 seconds for WikiTalk.) The algorithm by Jha et al. has much worse running time: above 10 minutes for Web-NotreDame and above 15 minutes for WikiTalk. This is due to the fact that for the first processed edges the probability that an edge is sampled is large.

6 Conclusions and future directions

We presented a new triangle counting algorithm for massive graphs revealed as a stream of edges in arbitrary order and showed that for graphs with a small vertex cover, our algorithm yields more accurate estimated than state-of-the-art approaches using same amount of space. It is easy to generalize the algorithm to counting k-cliques: a vertex cover must contain at least $k-1$ vertices from each k-clique and instead of 2-paths one will sample from the set of $(k-1)$-stars. (An ℓ-star is a connected graph with one internal vertex of degree ℓ and ℓ vertices of degree 1.) It is interesting to generalize our algorithm to more general subgraphs of fixed size, for example (k,t)-pseudocliques, i.e., k-cliques with at most t missing edges. Such graphs have been shown to emerge in various applications [19].

Acknowledgements. We would like to thank Rasmus Pagh for valuable comments. The work of the second author was done while visiting Yahoo Labs, Barcelona and was supported by the Danish National Research Council under the Sapere Aude program.

References

[1] N. Alon, R. Yuster, U. Zwick. Finding and Counting Given Length Cycles. *Algorithmica* 17(3): 209–223 (1997)

[2] H. Avron. Counting triangles in large graphs using randomized matrix trace estimation. *Large-Scale Data Mining: Theory and Applications (KDD Workshop)*, 2010.

[3] Z. Bar-Yossef, R. Kumar, D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. *SODA 2002*: 623–632

[4] L. Becchetti, P. Boldi, C. Castillo, A. Gionis. Efficient algorithms for large-scale local triangle counting. *TKDD 4(3)*: (2010)

[5] I. Bordino, D. Donato, A. Gionis, S. Leonardi. Mining Large Networks with Subgraph Counting. *ICDM 2008*: 737–742

[6] L. S. Buriol, G. Frahling, S. Leonardi, C. Sohler. Estimating Clustering Indexes in Data Streams. *ESA 2007*: 618–632

[7] L. S. Buriol, G. Frahling, S. Leonardi, A. Marchetti-Spaccamela, C. Sohler. Counting triangles in data streams. *PODS 2006*: 253–262

[8] R. Canetti, G. Even, O. Goldreich. Lower Bounds for Sampling Algorithms for Estimating the Average. *Inf. Process. Lett. 53(1)*: 17–25 (1995)

[9] L. Carter, M. N. Wegman. Universal Classes of Hash Functions. *J. Comput. Syst. Sci. 18(2)*: 143–154 (1979)

[10] J. Cheng, Z. Shang, H. Cheng, H. Wang, J. X. Yu. K-Reach: Who is in Your Small World. *PVLDB 5(11)*:1292-1303 (2012).

[11] S. Chu, J. Cheng. Triangle listing in massive networks. *TKDD 6(4): 17* (2012)

[12] D. Coppersmith, R. Kumar. An improved data stream algorithm for frequency moments. *SODA 2004*: 151–156

[13] C. Demetrescu, B. Escoffier, G. Moruz, A. Ribichini. Adapting parallel algorithms to the W-Stream model, with applications to graph problems. *Theor. Comput. Sci. 411(44-46):3994-4004* (2010)

[14] O. Green, D. A. Bader. Faster Clustering Coefficient Using Vertex Covers. *SocialCom 2013*: 321–330

[15] M. Henzinger, P. Raghavan, and S. Rajagopalan. Computing on data streams. *Technical Report TR 1998-011*, Compaq Systems Research Center, Palo Alto, CA, 1998

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

[17] I. Fudos, C. M. Hoffmann. A Graph-Constructive Approach to Solving Systems of Geometric Constraints. *ACM Trans. Graph. 16(2)*: 179–216 (1997)

[18] M. Jha, C. Seshadhri, A. Pinar. A space efficient streaming algorithm for triangle counting using the birthday paradox. *KDD 2013*: 589–597

[19] D. Jiang, J. Pei. Mining frequent cross-graph quasi-cliques. *TKDD 2(4)* (2009)

[20] H. Jowhari, M. Ghodsi. New Streaming Algorithms for Counting Triangles in Graphs. *COCOON 2005*: 710–716

[21] R. M. Karp (1972). Reducibility Among Combinatorial Problems. *Complexity of Computer Computations 1972*: 85–103

[22] M. N. Kolountzakis, G. L. Miller, R. Peng, C. E. Tsourakakis. Efficient Triangle Counting in Large Graphs via Degree-based Vertex Partitioning. *Internet Mathematics 8(1-2)*, 161–185 (2012)

[23] R. Kumar, P. Raghavan, S. Rajagopalan, A. Tomkins. Trawling the Web for Emerging Cyber-Communities. *Computer Networks* 31(11-16): 1481–1493 (1999)

[24] K. Kutzkov, R. Pagh. On the streaming complexity of computing local clustering coefficients. *WSDM 2013*: 677–686

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

[26] M. Pătraşcu, M. Thorup. The Power of Simple Tabulation Hashing. *J. ACM 59(3): 14* (2012)

[27] M. Ruhl. Efficient Algorithms for New Computational Models. *PhD thesis*, MIT (2003)

[28] C. Seshadhri, A. Pinar, and T. Kolda. Triadic Measures on Graphs: The Power of Wedge Sampling. *SDM 2013*: 10–18.

[29] C. E. Tsourakakis. Fast Counting of Triangles in Large Real Networks without Counting: Algorithms and Laws. *ICDM 2008*: 608–617

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

[31] C. E. Tsourakakis, M. N. Kolountzakis, G. L. Miller. Triangle Sparsifiers. *J. of Graph Algorithms and Appl. 15(6)*: 703–726 (2011)

[32] V. Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. *STOC 2012*, 887–898

[33] B. Viswanath, A. Mislove, M. Cha, P. K. Gummadi. On the evolution of user interaction in Facebook. *WOSN 2009*: 37–42

[34] D. J. Watts, S. H. Strogatz. Collective dynamics of "small world" networks. *Nature, 393:* 440–442, 1998.

[35] Y. Yao, J. Zhou, L. Han, F. Xu, J. Lu. Comparing Linkage Graph and Activity Graph of Online Social Networks. *SocInfo 2011*: 84–97

[36] S. Y. Yook, Z.N. Oltvai, and A.-L. Barabasi. Functional and topological characterization of protein interaction networks. *Proteomics, 4*, 928–942, 2004.