Triangle counting in dynamic graph streams[⋆]

Konstantin Kutzkov and Rasmus Pagh

IT University of Copenhagen, Denmark

Abstract. Estimating the number of triangles in graph streams using a limited amount of memory has become a popular topic in the last decade. Different variations of the problem have been studied, depending on whether the graph edges are provided in an arbitrary order or as incidence lists. However, with a few exceptions, the algorithms have considered *insert-only* streams. We present a new algorithm estimating the number of triangles in *dynamic* graph streams where edges can be both inserted and deleted. We show that our algorithm achieves better time and space complexity than previous solutions for various graph classes, for example sparse graphs with a relatively small number of triangles. Also, for graphs with constant transitivity coefficient, a common situation in real graphs, this is the first algorithm achieving constant processing time per edge. The result is achieved by a novel approach combining sampling of vertex triples and sparsification of the input graph.

1 Introduction

Many relationships between real life objects can be abstractly represented as graphs. The discovery of certain structural properties in a graph, which abstractly describes a given real-life problem, can often provide important insights into the nature of the original problem. The number of triangles, and the closely related clustering and transitivity coefficients, have proved to be an important measure used in applications ranging from social network analysis and spam detection to motif detection in protein interaction networks. We refer to [23] for a detailed discussion on the applications of triangle counting.

The best known algorithm for triangle counting in the RAM model runs in time $O(m^{\frac{2\omega}{\omega+1}})$ [4] where ω is the matrix multiplication exponent, the best known bound is $\omega = 2.3727$ [24]. However, this algorithm is mainly of theoretical importance since exact fast matrix multiplication algorithms do not admit an efficient implementation for input matrices of reasonable size.

The last decade has witnessed a rapid growth of available data. This has led to a shift in attitudes in algorithmic research and solutions storing the whole input in main memory are not any more considered a feasible choice for many real-life problems. Classical algorithms have been adjusted in order to cope with the new requirements and many new techniques have been developed [17].

⋆ This work is supported by the Danish National Research Foundation under the Sapere Aude program.

Approximate triangle counting in streamed graphs. For many applications one is satisfied with a good approximation of the number of triangles instead of their exact number, thus researchers have designed randomized approximation algorithms returning with high probability a precise estimate using only small amount of main memory. Two models of streamed graphs have been considered. In the *incidence list stream* model the edges incident to each vertex arrive consecutively and in the *adjacency stream* model edges arrive in arbitrary order. Also, it has been distinguished between algorithms using only a single pass over the input, and algorithms assuming that the input graph can be persistently stored on a secondary device and multiple passes are allowed. The one-pass algorithms with the best known space complexity and constant processing time per edge, both in the incidence list stream and adjacency stream model, are due to Buriol et al. [8], and when several passes are allowed – by Kolountzakis et al. [14]. For an overview of results and developed techniques we refer to [23].
Dynamic graph streams have a wider range of applications. Consider for example a social network like Facebook where one is allowed to befriend and "unfriend" other members, or join and leave groups of interest. Estimating the number of triangles in a network is a main building block in algorithms for the detection of emerging communities [7], and thus it is required that triangle counting algorithms can also handle edge deletions. The problem of designing triangle counting algorithms for dynamic streams matching the space and time complexity of algorithms for insert-only streams has been presented as an open question in the 2006 IITK Workshop on Algorithms for Data Streams [15]. The best known algorithms for insert-only streams work by sampling a non-empty subgraph on three vertices from the stream (e.g. an edge (u, v) and a vertex w). Then one checks whether the arriving edges will complete the sampled subgraph to a triangle (we look for (u, w) and (v, w)). The approach does not work for dynamic streams because an edge in the sampled subgraph might be deleted later. Proposed solutions [1, 16] have explored different ideas. These approaches, however, only partially resolve the open problem from [15] because of high processing time per edge update, see Section 3 for more details.

Our contribution. In this work we propose a method to adjust sampling to work in dynamic streams and show that for graphs with constant transitivity coefficient, a ubiquitous assumption for real-life graphs, we can achieve constant processing time per edge. At a very high level, the main technical contribution of the present work can be summarized as follows.
For dynamic graph streams sampling-based approaches fail because we don't know how many of the sampled subgraphs will survive after edges have been deleted. On the other hand, graph sparsification approaches [19, 22, 23] can handle edge deletions but the theoretical guarantees on the complexity of the algorithms depend on specific properties of the underlying graph, e.g., the maximum number of triangles an edge is part of. The main contribution in the present work is a novel technique for sampling 2-paths *after* the stream has been processed. It is based on the combination of standard 2-path sampling with graph sparsifica-

tion. The main technical challenge is to show that sampling at random a 2-path in a sparsified graph is (almost) equivalent to sampling at random a 2-path in the original graph. In the course of the analysis, we also obtain combinatorial results about general graphs that might be of independent interest.

2 Preliminaries

Notation. A simple undirected graph without loops is denoted as $G = (V, E)$ with $V = \{1, 2, \dots, n\}$ being a set of vertices and E a set of edges. The edges are provided as a stream of insertions and deletions in arbitrary order. We assume the strict turnstile model where each edge can be deleted only after being inserted. We assume that n is known in advance[1] and that the number of edges cannot exceed m. For an edge connecting the vertices u and v we write (u, v) and u and v are the *endpoints* of the edge (u, v). Vertex u is *neighbor* of v and vice versa and $N(u)$ is the set of u's neighbors. We say that edge (u, v) is *isolated* if $|N(u)| = |N(v)| = 1$. We consider only edges (u, v) with $u < v$. A *2-path centered at* v, (u, v, w), consists of the edges (u, v) and (v, w). A k-clique in G is a subgraph of G on k vertices $v_1, \dots, v_k$ such that $(v_i, v_j) \in E$ for all $1 \le i < j \le k$. A 3-clique on u, v, w is called a *triangle* on u, v, w, and is denoted as $\langle u, v, w \rangle$. We denote by $P_2(v)$ the number of 2-paths centered at a vertex v, and $P_2(G) = \sum_{v \in V} P_2(v)$ and $T_3(G)$ the number of 2-paths and number of triangles in G, respectively. We will omit G when clear from the context.

We say that two 2-paths are *independent* if they have at most one common vertex. The *transitivity coefficient* of G is

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

i.e., the ratio of 2-paths in G contained in a triangle to all 2-paths in G. When clear from the context, we will omit G.

Hashing. A family $\mathcal{F}$ of functions from U to a finite set S is *k-wise independent* if for a function $f : U \to S$ chosen uniformly at random from $\mathcal{F}$ it holds

$$\mathbf{Pr}[f(u_1) = c_1 \wedge f(u_2) = c_2 \wedge \cdots \wedge f(u_k) = c_k] = 1/s^k$$

for $s = |S|$, distinct $u_i \in U$ and any $c_i \in S$ and $k \in \mathbb{N}$. We will call a function chosen uniformly at random from a k-wise independent family *k-wise independent function* and a function $f : U \to S$ *fully random* if f is $|U|$-wise independent. We will say that a function $f : U \to S$ *behaves like a fully random function* if for any set of input from U, with high probability f has the same probability distribution as a fully random function.

We will say that an algorithm returns an *(ε, δ)-approximation* of some quantity q if it returns a value $\tilde{q}$ such that $(1 - \varepsilon)q \le \tilde{q} \le (1 + \varepsilon)q$ with probability at least $1 - \delta$ for every $0 < \varepsilon, \delta < 1$.

[1] Our results hold when the n vertices come from some arbitrary universe U and are known in advance. We omit this generalization due to lack of space.

3 The new approach

The following theorem is our main result.

Theorem 1 *Let $G = (V, E)$ be a graph given as a stream of edge insertions and deletions with no isolated edges and vertices, $V = \{1, 2, \ldots, n\}$ and $|E| \leq m$. Let P_2, T_3 and α be the number of 2-paths, number of triangles and the transitivity coefficient of G, respectively. Let $\varepsilon, \delta \in (0, 1)$ be user defined and $b = \max(n, P_2/n)$. Assuming fully random hash functions, there exists a one-pass algorithm running in expected space $O(\frac{m}{\sqrt{b}\varepsilon^3\alpha} \log \frac{1}{\delta})$ and $O(\frac{1}{\varepsilon^2\alpha} \log \frac{1}{\delta})$ processing time per edge. After processing the stream, an (ε, δ)-approximation of T_3 can be computed in expected time $O(\frac{\log n}{\varepsilon^2\alpha} \log \frac{1}{\delta})$ and worst case time $O(\frac{\log^2 n}{\varepsilon^2\alpha} \log \frac{1}{\delta})$ with high probability.*

(For simplicity, we assume that there are no isolated edges in G. More generally, the result holds by replacing n with n_C, where n_C is the number of vertices in connected components with at least two edges. We recall again that we assume m and n can be described in $O(1)$ words.)

	Space	Update time
Ahn et al.[1]	$O(\frac{mn}{\varepsilon^2 T_3} 1 \log \frac{1}{\delta})$	$O(n \log n)$
Manjunath et al. [16]	$O(\frac{m^3}{\varepsilon^2 T_3^2} \log \frac{1}{\delta})$	$O(\frac{m^3}{\varepsilon^2 T_3^2} \log \frac{1}{\delta})$
This work	$O(\frac{m}{\sqrt{b}\varepsilon^3\alpha} \log \frac{1}{\delta})$	$O(\frac{1}{\varepsilon^2\alpha} \log \frac{1}{\delta})$

Table 1. Overview of time and space bounds. It holds $b = \max(n, P_2/n)$.

	$n \log n$	m^3/T_3^2
$z < 1/2$	$T_3 = \omega(C^2/(n^{2z} \log n))$	$T_3 = o(Cn^{2-z})$
$1/2 < z < 1$	$T_3 = \omega(C^2/(n \log n))$	$T_3 = o(Cn^{3-3z})$
$z > 1$	$T_3 = \omega(C^2/(n \log n))$	$T_3 = o(C)$

Table 2. Comparison of the theoretical guarantees for the per edge processing time for varying z.

Before presenting the algorithm, let us compare the above to the bounds in [1, 16]. The algorithm in [1] estimates T_3 by applying ℓ_0 sampling [11] to non-empty subgraphs on 3 vertices. There are $O(mn)$ such subgraphs, thus $O(\frac{mn}{\varepsilon^2 T_3} \log \frac{1}{\delta})$ samples are needed for an (ε, δ)-approximation. However, each edge insertion or deletion results in the update of $n - 2$ non-empty subgraphs on 3 vertices. Using the ℓ_0 sampling algorithm from [12], this results in processing time of $O(n \log n)$ per edge. The algorithm by Manjunath et al. [16] estimates the number of triangles (and more generally of cycles of fixed length) in streamed graphs by computing complex valued sketches of the stream. Each of them yields an unbiased estimator of T_3. The average of $O(\frac{m^3}{\varepsilon^2 T_3^2} \log \frac{1}{\delta})$ estimators is an (ε, δ)-approximation of T_3. However, each new edge insertion or deletion has to update all estimators, resulting in update time of $O(\frac{m^3}{\varepsilon^2 T_3^2} \log \frac{1}{\delta})$. The algorithm was generalized to counting arbitrary subgraphs of fixed size in [13].

The time and space bounds are summarized in Table 1. Comparing our space complexity to the bounds in [1, 16], we see that for several graph classes our algorithm is more time and space efficient. (We ignore ε and δ and logarithmic factors in n for the space complexity.) For d-regular graphs the processing time per edge is better than $O(n \log n)$ for $T_3 = \omega(d^2/\log n)$, and better than $O(m^3/T_2^3)$ for $T_3 = o(n^2 d)$. Our space bound is better than $O(mn/T_3)$ when $d = o(n^{1/4})$, and better than $O(m^3/T_3^2)$ for $T_3 = o(\max(n^{3/2}, nd))$. Most real-life graphs exhibit a skewed degree distribution adhering to some form of power law, see for example [2]. Assume vertices are sorted according to their degree in decreasing order such that the ith vertex has degree C/i^z for some $C \leq n$, and constant $z > 0$, i.e., we have Zipfian distribution with parameter z. It holds $\sum_{i=1}^{n} i^{-z} = O(n^{1-z})$ for $z < 1$ and $\sum_{i=1}^{n} i^{-z} = O(1)$ for $z > 1$. Table 2 summarizes for which values of T_3 our algorithm achieves faster processing time than [1, 16], and Table 3 – for which values of C our algorithm is more space-efficient than [1], and for which values of T_3 – more space-efficient than [16].

However, the above values are for arbitrary graphs adhering to a certain degree distribution. We consider the main advantage of the new algorithm to be that it achieves constant processing time per edge for graphs with constant transitivity coefficient. This is a common assumption for real-life networks, see for instance [3, 8]. Note that fast update is essential for real life applications. Consider for example the Facebook graph. In May 2011, for less than eight years existence, there were about 69 billion friendship links [6]. This means an average of above 300 new links per second, without counting deletions and peak hours.

In the full version of the paper [2] we compare the theoretical guarantees for several real life graphs. While such a comparison is far from being a rigorous experimental evaluation, it clearly indicates that the processing time per edge in [1, 16] is prohibitively large and the assumption that the transitivity coefficient is constant is justified. Also, for graphs with a relatively small number of triangles our algorithm is much more space-efficient.

	mn/T_3	m^3/T_3^2
$z < 1/2$	$C = o(n^{1/4+z})$	$T_3 = o(\max(n^{3/2}, Cn^{1-z}))$
$1/2 < z < 1$	$C = o(n^{3/4})$	$T_3 = o(n^{5/2-2z})$
$z > 1$	$C = o(n^{3/4})$	$T_3 = o(n^{1/2})$

Table 3. Comparison of the theoretical guarantees for the space usage for varying z.

3.1 The main idea

The main idea behind our algorithm is to design of a new sampling technique for dynamic graph streams. It exploits a combination of the algorithms by Buriol et al. [8] for the incidence stream model, and the Doulion algorithm [22] and its improvement [19]. Let us briefly describe the approaches.

The Buriol et al. algorithm for incidence list streams. Assume we know the total number of 2-paths in G. One chooses at random one of them, say

[2] http://arxiv.org/pdf/1404.4696.pdf

(u, v, w), and checks whether the edge (u, w) appears later in the stream. For a triangle $\langle u, v, w \rangle$ the three 2-paths (u, v, w), (w, u, v), (v, w, u) appear in the incidence list stream, thus the probability that we sample a triangle is exactly α. One chooses independently at random K 2-paths and using standard techniques shows that for $K = O(\frac{1}{\varepsilon^2 \alpha} \log \frac{1}{\delta})$ we compute an (ε, δ)-approximation of $\alpha(G)$. One can get rid of the assumption that the number of 2-paths is known in advance by running $O(\log n)$ copies of the algorithm in parallel, each guessing the right value. The reader is referred to the original work for more details. For incidence streams, the number of 2-paths in G can be computed exactly by updating a single counter, thus $\tilde{T_3} = \tilde{\alpha} P_2$ is an (ε, δ)-approximation of T_3.

Doulion and monochromatic sampling. The Doulion algorithm [22] is a simple and intuitive sparsification approach. Each edge is sampled independently with probability p and added to a sparsified graph G_S. We expect pm edges to be sampled and a triangle survives in G_S with probability p^3, thus multiplying the number of triangles in G_S by $1/p^3$ we obtain an estimate of T_3. The algorithm was improved in [19] by using *monochromatic sampling*. Instead of throwing a biased coin for each edge, we uniformly at random color each vertex with one of $1/p$ colors. Then we keep an edge in the sparsified graph iff its endpoints have the same color. A triangle survives in G_S with probability p^2. It is shown that for a fully random coloring the variance of the estimator is better than in Doulion. However, in both algorithms it depends on the maximum number of triangles an edge is part of, and one might need constant sampling probability in order to obtain an (ε, δ)-approximation on T_3. The algorithm can be applied to dynamic streams because one counts the number of triangles in the sparsified graph after all edges have been processed. However, it can be expensive to obtain an estimate since the exact number of triangles in G_S is required.

Combining the above approaches. The basic idea behind the new algorithm is to use the estimator of Buriol et al. for the incidence stream model: (i) estimate the transitivity coefficient $\alpha(G)$ by choosing a sufficiently large number of 2-paths at random and check which of them are part of a triangle, and (ii) estimate the number of 2-paths P_2 in the graph. We first observe that estimating P_2 in dynamic graph streams can be reduced to second moment estimation of streams of items in the turnstile model, see e.g. [21]. For (i), we will estimate $\alpha(G)$ by adjusting the monochromatic sampling approach. Its main advantage compared to the sampling of edges separately is that if we have sampled the 2-path (u, v, w), then we must also have sampled the edge (u, w), if existent. So, the idea is to use monochromatic sampling and then in the sparsified graph to pick up at random a 2-path and check whether it is part of a triangle. Instead of random coloring of the vertices, we will use a suitably defined hash function and we will choose a sampling probability guaranteeing that for a graph with no isolated edges (or rather a small number of isolated edges) the sparsified graph will contain a sufficiently big number of 2-paths. A 2-path in the sparsified graph picked up at random, will then be used to estimate $\alpha(G)$. Thus, unlike in [8], we sample *after* the stream has been processed and this allows to handle edge deletions. The main technical obstacles are to analyze the required sampling probability p and to show that this sampling approach indeed provides an unbiased estimator of $\alpha(G)$. We will obtain bounds on p and show that even if the estimator might be biased, the bias can be made arbitrarily small and one can still achieve an (ε, δ)-approximation of $\alpha(G)$. Also, we present an implementation for storing a sparsified graph G_S such that each edge is added or deleted in constant time and a random 2-path in G_S, if existent, can be picked up without explicitly considering all 2-paths in G_S.

3.2 The algorithm

Pseudocode description of the algorithm is given in Figure 1. We assume that the graph is given as a stream $\mathcal{S}$ of pairs $((u, v), \$)$, where $(u, v) \in E$ and $\$ \in \{+, -\}$ with the obvious meaning that the edge (u, v) is inserted or deleted from G. In EstimateNumberOfTwoPaths each incoming pair $((u, v), \$)$ is treated as the insertion, respectively deletion, of two items u and v, and these update a second moment estimator SME, working as a blackbox algorithm. We refer to the proof of Lemma 1 for more details. In SparsifyGraph we assume access to a fully random coloring hash function $f : V \to C$. Each edge (u, v) is inserted/deleted to/from a sparsified graph G_S iff $f(u) = f(v)$. At the end G_S consists of all monochromatic edges that have not been deleted. In EstimateNumberOfTriangles we run in parallel the algorithm estimating P_2 and K copies of SampleRandom2Path. For each G_S^i, $1 \leq i \leq K$, with at least s pairwise independent 2-paths we choose at random a 2-path and check whether it is a triangle. (Note that we require the existence of s *pairwise independent* 2-paths but we choose a 2-path at random from *all* 2-paths in G_S.) The ratio of triangles to all sampled 2-paths and the estimate of P_2 are then used to estimate T_3. In the next section we obtain bounds on the user defined parameters C, K and s. In Lemma 6 we present en efficient implementation of G_S that guarantees constant time updates and allows the sampling of a random 2-path in expected time $O(\log n)$ and worst case time $O(\log^2 n)$ with high probability.

3.3 Theoretical analysis

We will prove the main result in several lemmas. Due to lack of space, proofs which are not essential for the understanding of the main ideas can be found in the full version of the paper. The next lemma provides an estimate of P_2 using an estimator for the second frequency moment of data streams [21].

Lemma 1 *Let G be a graph with no isolated edges given as a stream of edge insertions and deletions. There exists an algorithm returning an (ε, δ)-approximation of the number of 2-paths in G in one pass over the stream of edges which needs $O(\frac{1}{\varepsilon^2} \log \frac{1}{\delta})$ space and $O(\log \frac{1}{\delta})$ processing time per edge.*

The next two lemmas will show a lower bound on the number of pairwise independent 2-paths in a graph without isolated edges. The results are needed in

```
EstimateNumberOfTwoPaths
Input: stream of edge deletions and insertions $\mathcal{S}$, algorithm $SME$ estimating the
    second moment items streams
1: $m = 0$
2: for each $((u, v), \$)$ in $\mathcal{S}$ do
3:     if $\$ = +$ then
4:         $m = m + 1$
5:         $SME.update(u, 1)$, $SME.update(v, 1)$
6:     else
7:         $m = m - 1$
8:         $SME.update(u, -1)$, $SME.update(v, -1)$
9: return $SME.estimate/2 - m$
```

```
SparsifyGraph
Input: stream of edge deletions and insertions $\mathcal{S}$, coloring function $f : V \to C$
1: $G_S = \emptyset$
2: for each $((u, v), \$) \in \mathcal{S}$ do
3:     if $f(u) = f(v)$ then
4:         if $\$ = +$ then
5:             $G_S = G_S \cup (u, v)$.
6:         else
7:             $G_S = G_S \backslash (u, v)$.
8: Return $G_S$.
```

```
SampleRandom2Path
Input: sparsified graph $G_S$
1: choose at random a 2-path $(u, v, w)$ in $G_S$
2: if the vertices $\{u, v, w\}$ form a triangle then
3:     return 1
4: else
5:     return 0
```

```
EstimateNumberOfTriangles
Input: streamed graph $\mathcal{S}$, set of $K$ independent fully random coloring functions $\mathcal{F}$,
    algorithm $SME$ estimating the second moment of streams of items, threshold $s$
1: run in parallel EstimateNumberOfTwoPaths($\mathcal{S}, SME$) and let $\tilde{P}_2$ be the re-
   turned estimate
2: run in parallel $K$ copies of SparsifyGraph($S, f_i$), $f_i \in \mathcal{F}$
3: $\ell = 0$
4: for each $G_S^i$ with at least $s$ pairwise independent 2-paths do
5:     $X += $ SampleRandom2Path($G_S^i$)
6:     $\ell += 1$
7: $\tilde{\alpha} = X/\ell$
8: return $\frac{\tilde{\alpha}\tilde{P}_2}{3}$
```

Fig. 1. Estimating the number of 2-paths in G, the transitivity coefficient and the number of triangles.

order to obtain bounds on the required sampling probability. First we show that a graph without isolated edges contains a linear number of pairwise independent 2-paths.

Lemma 2 *Let $G = (V, E)$ be a graph over n vertices without isolated edges. Then there exist at least $\Omega(n)$ pairwise independent 2-paths.*

The next result gives a lower bound on the number of pairwise independent 2-paths in terms of the total number of 2-paths. For denser graphs it implies the existence of $\omega(n)$ pairwise independent 2-paths.

Lemma 3 *Let the number of 2-paths in a graph $G = (V, E)$ be P_2. There exist $\Omega(P_2/n)$ pairwise independent 2-paths.*

Next we obtain bounds on the sampling probability such that there are sufficiently many pairwise independent 2-paths in G_S. As we show later, this is needed to guarantee that SampleRandom2Path will return an almost unbiased estimator of the transitivity coefficient. The events for two 2-paths being monochromatic are independent, thus the next lemma follows from Lemma 2 and Chebyshev's inequality. Note that we still don't need the coloring function f to be fully random.

Lemma 4 *Let f be 6-wise independent and $p \geq \frac{5\sqrt{3}}{\varepsilon\sqrt{b}}$ for $b = \max(n, P_2/n)$ and $\varepsilon \in (0, 1]$. Then with probability at least 3/4* SparsifyGraph *returns G_S such that there are at least $18/\varepsilon^2$ pairwise independent 2-paths in G_S.*

Lemma 5 *Assume we run* EstimateNumberOfTriangles *with $s = 18/\varepsilon^2$ and let X be the value returned by* SampleRandom2Path. *Then $(1-\varepsilon)\alpha \leq \mathbb{E}[X] \leq (1+\varepsilon)\alpha$.*

Proof. We analyze how much differs the probability between 2-paths to be selected by SampleRandom2Path. Consider a given 2-path (u, v, w). It will be sampled if the following three events occur:

1. (u, v, w) is monochromatic, i.e., it is in the sparsified graph G_S.
2. There are $i \geq 18/\varepsilon^2$ pairwise independent 2-paths in G_S.
3. (u, v, w) is selected by SampleRandom2Path.

The first event occurs with probability p^2. Since f is fully random, the condition that (u, v, w) is monochromatic does not alter the probability for any 2-path independent from (u, v, w) to be also monochromatic. The probability to be in G_S changes only for 2-paths containing two vertices from $\{u, v, w\}$, which in turn changes the number of 2-paths in G_S and thus probability for (u, v, w) to be picked up by SampleRandom2Path. In the following we denote by p_{G_S} the probability that a given 2-path is monochromatic and there are at least $18/\varepsilon^2$ pairwise independent 2-paths in G_S, note that p_{G_S} is equal for all 2-paths.

Consider a fixed coloring to $V \backslash \{u, v, w\}$. We analyze the difference in the number of monochromatic 2-paths depending whether $f(u) = f(v) = f(w)$ or not. There are two types of 2-paths that can become monochromatic conditioning on $f(u) = f(v) = f(w)$: either (i) 2-paths with two endpoints in $\{u, v, w\}$ centered at some $\{u, v, w\}$, or (ii) 2-paths with two vertices in $\{u, v, w\}$ centerer at a vertex $x \notin \{u, v, w\}$. For the first case assume w.l.o.g. there is a 2-path $(u, v, w) \in G_S$ centered at v and let $d_v^{f(v)} = |\{z \in N(v)\backslash\{u, w\} : f(z) = f(v)\}|$, i.e., $d_v^{f(v)}$ is the number of v's neighbors different from u and w, having the same color as v. Thus, the number of monochromatic 2-paths centered at v varies by $2d_v^{f(v)}$ conditioning on the assumption that $f(u) = f(v) = f(w)$. The same reasoning applies also to the 2-paths centered at u and w. For the second case consider the vertices u and v. Conditioning on $f(u) = f(w)$, we additionally add to G_S 2-paths (u, x_i, w) for which $f(x_i) = f(u) = f(w)$ and $x_i \in N(u) \cap N(w)$. The number of such 2-paths is at most $\min(d_u^{f(u)}, d_w^{f(w)})$. The same reasoning applies to any pair of vertices from $\{u, v, w\}$. Therefore, depending on whether $f(u) = f(v) = f(w)$ or not, the number of monochromatic 2-paths centered at a vertex from $\{u, v, w\}$ varies between

$$\sum_{y\in\{u,v,w\}} \binom{d_y^{f(y)}}{2} \qquad \text{and} \qquad \sum_{y\in\{u,v,w\}} \binom{d_y^{f(y)}}{2} + 3d_y^{f(y)}.$$

Set $k = 18/\varepsilon^2$. Consider now two different, but not necessarily independent, 2-paths $(u_1, v_1, w_1), (u_2, v_2, w_2) \in G$. We analyze the probability for each of them to be selected by SampleRandom2Path. Let $\mathcal{C}$ be a partial coloring to $V\backslash\{u_j, v_j, w_j\}$, $j = 1, 2$. If $\mathcal{C}$ is completed to a coloring of all vertices such that both (u_1, v_1, w_1) and (u_2, v_2, w_2) are monochromatic, then clearly they are picked up with the same probability. Assume that with probability p_i, $i-1$ 2-paths are colored monochromatic by $\mathcal{C}$ and consider extensions of $\mathcal{C}$ that make exactly one of (u_1, v_1, w_1) and (u_2, v_2, w_2) monochromatic. Under the assumption there are $i \geq k$ 2-paths in G_S and following the above discussion about the number of 2-paths with at least two vertices from $\{u_j, v_j, w_j\}$, we see that the number of monochromatic 2-paths can vary between i and $i + 3\sqrt{2i}$. Thus, the probability for (u_1, v_1, w_1) and (u_2, v_2, w_2) to be sampled varies between

$$p_{G_S} \sum_{i\geq k} \frac{p_i}{i} \qquad \text{and} \qquad p_{G_S} \sum_{i\geq k} \frac{p_i}{i + 3\sqrt{2i}}.$$

We assume G_S contains at least k 2-paths, thus $\sum_{i\geq k} p_i = 1$ and there exists $r \geq k, r \in \mathbb{R}$ such that $\sum_{i\geq k} p_i i^{-1} = 1/r$. Thus we bound

$$\sum_{i\geq k} \frac{p_i}{i + 3\sqrt{2i}} = \sum_{i\geq k} \frac{p_i}{i(1 + 3\sqrt{2/i})} \geq \frac{1}{1 + 3\sqrt{2/k}} \sum_{i\geq k} \frac{p_i}{i} = \frac{1}{r(1 + 3\sqrt{2/k})}.$$

Since the function f is fully random, each coloring is equally probable. The above reasoning applies to any pair of 2-paths in G, thus for any 2-path the probability to be sampled varies between

$$\frac{p_{G_S}}{r} \qquad \text{and} \qquad \frac{p_{G_S}}{(1 + \sqrt{18/k})r} = \frac{p_{G_S}}{(1 + \varepsilon)r}.$$

Assume first the extreme case that 2-paths which are not part of a triangle are sampled with probability $\frac{1}{r}$ and 2-paths part of a triangle with probability $\frac{1}{(1+\varepsilon)r}$. We have $X = \sum_{(u,v,w)\in P_2} I_{(u,v,w)}$, where $I_{(u,v,w)}$ is an indicator random variable denoting whether (u, v, w) is part of a triangle. Thus

$$\mathbb{E}[X] \geq \frac{p_{G_S} 3T_3}{(1+\varepsilon)r} \frac{r}{p_{G_S} P_2} = \frac{\alpha}{1+\varepsilon} \geq (1-\varepsilon)\alpha.$$

On the other extreme, assuming that we select 2-paths part of triangles with probability $\frac{1}{r}$ and 2-paths not part of a triangle with probability $\frac{1}{r(1+\varepsilon)}$, using similar reasoning we obtain $\mathbb{E}[X] \leq (1+\varepsilon)\alpha$. □

Applying a variation of rejection sampling, in the next lemma we show how to store a sparsified graph G_S such that we efficiently sample a 2-path uniformly at random and G_S is updated in constant time.

Lemma 6 *Let $G_S = (V, E_S)$ be a sparsified graph over m' monochromatic edges. There exists an implementation of G_S in space $O(m')$ such that an edge can be inserted to or deleted from G_S in constant time with high probability. A random 2-path, if existent, can be selected from G_S in expected time $O(\log n)$ and $O(\log^2 n)$ time with high probability.*

Now we have all components in order to prove the main result.

Proof. (of Theorem 1).

Assume EstimateNumberOfTriangles runs K copies in parallel of SparsifyGraph with $p = \frac{5\sqrt{3}}{\varepsilon\sqrt{b}}$ for $b = \max(n, P_2/n)$. By Lemma 4 with probability 3/4 we have a sparsified graph with at least $s = 18/\varepsilon^2$ pairwise independent 2-paths. Thus, we expect to obtain from $3K/4$ of them an indicator random variable. A standard application of Chernoff's inequality yields that with probability $O(2^{-K/36})$ we will have $\ell \geq K/2$ indicator random variables X_i denoting whether the sampled 2-path is part of a triangle. By Lemma 5 we have $(1-\varepsilon)\alpha \leq \mathbb{E}[X_i] \leq (1+\varepsilon)\alpha$ and as an estimate of α we return $\sum_{i=1}^{\ell} X_i/\ell$. Observe that $(1+\varepsilon/3)^2 \leq 1+\varepsilon$, respectively $(1-\varepsilon/3)^2 \geq 1-\varepsilon$. From the above discussion and applying Chernoff's inequality and the union bound, we see that for $K = \frac{36}{\varepsilon^2\alpha}\log\frac{2}{\delta}$, we obtain an $(\varepsilon, \delta/2)$-approximation of α.

By Lemma 1 we can compute an $(\varepsilon, \delta/2)$-approximation of the number of 2-paths in space $O(\frac{1}{\varepsilon^2}\log\frac{1}{\delta})$ and $O(\log\frac{1}{\delta})$ per edge processing time. It is trivial to show that this implies an $(3\varepsilon, \delta)$-approximation of the number of triangles for $\varepsilon < 1/3$. Clearly, one can rescale ε in the above, i.e. $\varepsilon = \varepsilon/3$, such that EstimateNumberOfTriangles returns an (ε, δ)-approximation.

By Lemma 6, each sparsified graph with m' edges uses space $O(m')$ and each update takes constant time with high probability, thus we obtain that each edge is processed with high probability in time $O(K)$. Each monochromatic edge and its color can be represented in $O(\log n)$ bits.

By Lemma 6, in expected time $O(\log n)$ and worst case time $O(\log^2 n)$ with high probability we sample uniformly at random a 2-path from each G_S with at least $18/\varepsilon^2$ pairwise independent 2-paths. □

References

1. K. J. Ahn, S. Guha, A. McGregor. Graph sketches: sparsification, spanners, and subgraphs. *PODS 2012*: 5–14
2. W. Aiello, F. R. K. Chung, L. Lu. A random graph model for massive graphs. *STOC 2000*: 171–180
3. R. Albert, A.-L. Barabási. Statistical mechanics of complex networks. *Rev. Mod. Phys. 74*, 47–97 (2002)
4. N. Alon, R. Yuster, U. Zwick. Finding and Counting Given Length Cycles. *Algorithmica* 17(3): 209–223 (1997)
5. Y. Arbitman, M. Naor, G. Segev. Backyard Cuckoo Hashing: Constant Worst-Case Operations with a Succinct Representation. *FOCS 2010*: 787–796
6. L. Backstrom, P. Boldi, M. Rosa, J. Ugander, S. Vigna. Four degrees of separation. *WebSci 2012*: 33–42
7. J.W. Berry, B. Hendrickson, R. LaViolette, C.A. Phillips. Tolerating the Community Detection Resolution Limit with Edge Weighting. *Phys. Rev. E, 83(5)*
8. L. S. Buriol, G. Frahling, S. Leonardi, A. Marchetti-Spaccamela, C. Sohler. Counting triangles in data streams. *PODS 2006*: 253–262
9. L. Carter, M. N. Wegman. Universal Classes of Hash Functions. *J. Comput. Syst. Sci. 18(2)*: 143–154 (1979)
10. M. Dietzfelbinger. Design Strategies for Minimal Perfect Hash Functions. *SAGA 2007*: 2–17
11. G. Frahling, P. Indyk, C. Sohler. Sampling in dynamic data streams and applications. *Symposium on Computational Geometry 2005*: 142–149
12. H. Jowhari, M. Saglam, G. Tardos. Tight bounds for Lp samplers, finding duplicates in streams, and related problems. *PODS 2011*: 49–58
13. D. M. Kane, K. Mehlhorn, T. Sauerwald, H. Sun. Counting Arbitrary Subgraphs in Data Streams. *ICALP (2) 2012*: 598–609
14. 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)
15. S. Leonardi. List of Open Problems in Sublinear Algorithms: Problem 11. http://sublinear.info/11
16. M. Manjunath, K. Mehlhorn, K. Panagiotou, H. Sun. Approximate Counting of Cycles in Streams. *ESA 2011*: 677–688
17. S. Muthukrishnan. Data Streams: Algorithms and Applications. *Foundations and Trends in Theoretical Computer Science*, Vol. 1, Issue 2, 2005
18. A. Pagh, R. Pagh. Uniform Hashing in Constant Time and Optimal Space. *SIAM J. Comput. 38(1)*: 85–96 (2008)
19. R. Pagh, C. E. Tsourakakis. Colorful triangle counting and a MapReduce implementation. *Inf. Process. Lett. 112(7):* 277–281 (2012)
20. M. Pătraşcu, M. Thorup. The Power of Simple Tabulation Hashing. *J. ACM 59(3): 14* (2012)
21. M. Thorup, Y. Zhang. Tabulation based 4-universal hashing with applications to second moment estimation. *SODA 2004*: 615–624
22. C. E. Tsourakakis, U. Kang, G. L. Miller, C. Faloutsos. DOULION: counting triangles in massive graphs with a coin. *KDD 2009*: 837–846
23. C. E. Tsourakakis, M. N. Kolountzakis, G. L. Miller. Triangle Sparsifiers. *J. of Graph Algorithms and Appl. 15(6)*: 703–726 (2011)
24. V. Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. *STOC 2012*, 887–898