SAND2014-18716C

Counting Triangles in Real-World Graph Streams: Dealing with Repeated Edges and Time Windows *

Madhav Jha C. Seshadhri Ali Pinar †

Abstract

Real-world graphs often manifest as a massive temporal "stream" of edges. The need for real-time analysis of such large graph streams has led to progress on low memory, one-pass streaming graph algorithms. These algorithms were designed for simple graphs, assuming an edge is not repeated in the stream. Real graph streams however, are almost always multigraphs i.e., they contain many duplicate edges. The assumption of no repeated edges requires an extra pass *storing all the edges* just for deduplication, which defeats the purpose of small memory algorithms.

We describe an algorithm, MG-Triangle, for estimating the triangle count of a multigraph stream of edges. We show that all previous streaming algorithms for triangle counting fail for multigraph streams, despite their impressive accuracies for simple graphs. The bias created by duplicate edges is a major problem, and leads these algorithms astray. MG-Triangle avoids these biases through careful debiasing strategies and has provable theoretical guarantees and excellent empirical performance. MG-Triangle builds on the previously introduced wedge sampling methodology. Another challenge in analyzing temporal graphs is finding the right temporal window size. MG-Triangle seamlessly handles multiple time windows, and does not require committing to any window size(s) a priori. We apply MG-Triangle to discover fascinating transitivity and triangle trends in real-world graph streams.

1 Introduction

Many massive graphs appear in practice as a temporal *stream of edges*. People call each other on the phone, exchange emails, or co-author a paper; computers exchange messages; animals come in the vicinity of each other; companies trade with each other. Each such interaction is modeled as an edge in the graph, and has a natural timestamp.

Due to the need for real-time awareness despite the volume of such transactions, there is much interest in processing temporal graphs using fast, limited-memory algorithms. Formally, think of the input as a sequence of edges $e_1, e_2, \ldots, e_m$. Some of the edges may be repeated, meaning that (say) $e_1 = e_{100} = e_{125} = (u, v)$. We are interested in small space streaming algorithms that make a *single pass* over the stream $e_1, e_2, \ldots, e_m$. Such an algorithm maintains data structures that are many orders of magnitude smaller than the stream itself. At every timestep t, these data structures are updated rapidly (possible randomly). The algorithm computes an accurate estimate for the property of interest on the graph seen so far. Because of the single pass and small space, the algorithm cannot revisit edges that it has forgotten. Furthermore, it cannot always determine if the new edge, e_t, has appeared before. This work focuses on triangle counting in this setting.

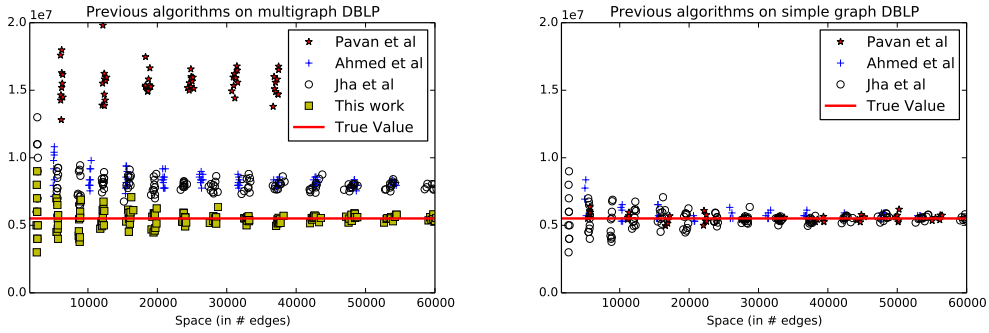
Graph vs multigraph: Previous results assume that the edge stream forms a simple graph, and no edge is repeated in the stream. This is a useful assumption for algorithmic progress; yet, often false in practice. Real-world graph streams are multigraphs, in that same edges can occur repeatedly in the data stream. The simple graph representation is obtained by removing duplicate edges. For example, the classic `Enron` email dataset is really a multigraph with 1.14M edges, while the underlying simple graph has only 297K edges. Similarly, a `DBLP` co-authorship graph recently collected is a multigraph with 3.63M edges, but the underlying simple graph has only 2.54M edges. Close to 10 million edges in a popular dynamic `Flickr` network dataset (see [22, 19]) are repeated.

The assumption of simplicity is implemented in practice with an extra pass to remove duplicate edges. *This pass requires storage of the entire simple graph, which is completely ignored in all previous work.* Indeed, if one can store the entire simple graph, there exist much better algorithms for triangle counting [23, 29, 25]. We posit that for streaming algorithms to be actually useful in practice, multiple edges must be dealt with small space. There is much work on streaming graph algorithms (see surveys [1, 17]). Yet this algorithmic work ignores important issues such as repeated edges and temporal aggregation that arise when looking at a real-world graph stream, as demonstrated in Fig. 1a.

Aggregation over time: Given a stream of edges, what is the actual graph? The most common answer is to simply aggregate all edges ever seen. Again, this is a

*This work was funded by the DARPA GRAPHS and DOE ASCR applied math programs. Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

†Sandia National Laboratories, Livermore, CA. Email: {mjha, scomand, apinar}@sandia.gov;



(a) Real stream *(b)* Simple

Figure 1: Previous works when run on multigraph DBLP converge (as storage increases) to an incorrect value. On the simple graph DBLP, they converge correctly.

useful assumption for algorithmic progress, but ignores the temporal aspect of the edges. Time is a complex issue and there are no clear solutions. One may consider sliding windows in time or have some decay of edges. For simplicity, we focus on sliding time windows (like edges seen in the past month, or past year). Even for sliding windows, it is not clear what the width should be. Observations can often be an artifact of the window size [16]. Therefore, it is essential to observe multiple time windows at the same time, instead of committing to a single one.

Fig. 2 shows how our algorithm MG-Triangle can analyze different time windows with a single run. Our algorithm estimates the triangle counts for any time window without altering its data structures, as the time window is only used in calculating the estimate.

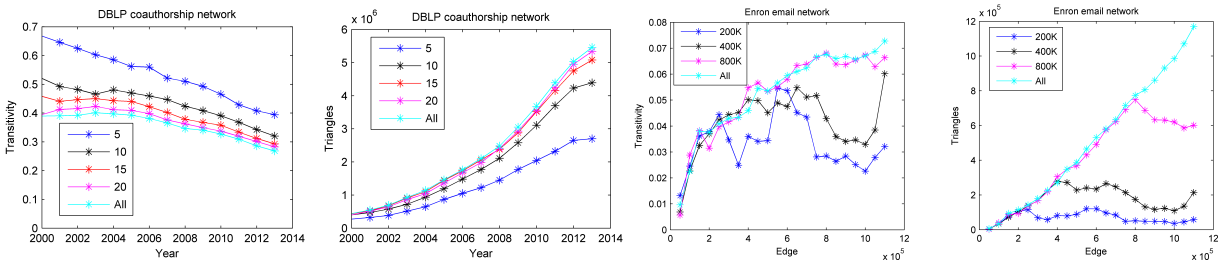
1.1 Triangle counting The abundance of triangles has been observed in networks arising in numerous scenarios, such as social sciences [7, 21, 6, 30], spam detection [10], community detection [11], finding common topics on the web [4], bioinformatics [18], and modeling and characterizing real-world networks [24, 9]. Subsequently, there has been a lot of work on triangle counting in graph streams [13, 5, 3, 14, 28, 20, 12, 2], and in various other settings (see e.g., [26] and references therein). The result of Ahmed et al. [2] is arguably the state-of-the-art, with a storage significantly smaller than previous algorithms. None of these results explicitly deal with multigraphs.

Formally, we are processing a multigraph stream $e_1, e_2, \ldots, e_m$. At every time t, consider the underlying *simple* graph G_t formed by edges $e_{t-\Delta t}, \ldots, e_t$. So take all these edges, and remove duplicates. We wish to output the triangle count (alternately, the transitivity) of G_t for all times t. The window length Δt may be defined in different ways. It could either be in terms of number of edges (say, the past 10K edges), or in terms of the semantics of timestamps (say, edges seen in the past month). Most importantly, we want a single-pass small space algorithm to handle multiple windows lengths and do not want different passes for each window length.

A reader may wonder why we only output estimates for the underlying simple graph. Ideally, we would like to compute measures that involve the multigraph structure. We agree that this is an interesting problem, and duplicates have their own significance. Currently, it is standard to focus on simple graphs, and there is no consensus on how to define triadic measures on multigraphs. This is an exciting avenue for future work.

Why is this a difficult problem? Multigraphs are a major challenge for triangle counting algorithms. Edges appears with varying frequencies, and (in our setting) we do not wish to be biased by this. Furthermore, triangles can be formed in different ways. Consider edges a, b, and c that form a triangle. These edges may appear in the multigraph stream in many different ways. For example, these edges could come as $a, a, \ldots, b, b, \ldots, c, c, \ldots$, or as $a, b, c, a, b, c, a, b, c, \ldots$. (Observe how this is *not* an issue for simple graphs.) These patterns create biases for existing triangle counting algorithms, which we explain in more detail later.

For now, it suffices to say that existing algorithms [5, 12, 20, 28, 2] will give different estimates for triangle counts of different multigraphs streams that contain the same simple graph. This is demonstrated in Fig. 1a, where we run previous streaming triangle counting algorithms on the raw DBLP multigraph stream. Previous algorithms converge to an incorrect value as their storage increases. They all perform extremely well if all duplicates were removed from the stream (Fig. 1b). Previous work on multigraph mining explicitly states



(a) DBLP: transitivity *(b)* DBLP: triangles *(c)* Enron: Transitivity *(d)* Enron: triangles

Figure 2: Transitivity and triangles estimates for varying window lengths for DBLP coauthorship and Enron email networks: Our algorithm stores less than 3% of the multigraph stream yet produces fine-grained triangle results.

triangle counting of streaming multigraphs as an open problem [8].

1.2 Preliminaries The edge stream is denoted by $e_1, e_2, \ldots, e_m$. We focus on undirected graphs, so each edge is an unordered pair of vertex ids. The simple graph formed by edges $e_{t'}, \ldots, e_t$ is denoted by $G[t', t]$. A *wedge* is a path of length 2. The set of wedges in a simple graph G is denoted $W(G)$, and the set of triangles by $T(G)$. A wedge in $W(G)$ is *closed* if it participates in a triangle and *open* otherwise. The *transitivity* is the fraction of closed wedges, $\tau(G) = 3|T(G)|/|W(G)|$. Our aim is to maintain the transitivity and triangle count (for all t) of the graph $G[t - \Delta t, t]$, where Δt is the desired window of aggregation. The window is usually specified as a fixed number of edges or a fixed interval of time (like month, year, etc.), though the algorithm works for windows lengths that change with time. For convenience, we denote $G_t = G[t-\Delta t, t]$, $E_t = E(G_t)$, $W_t = W(G_t)$, $T_t = T(G_t)$, and $\tau_t = \tau(G_t)$.

1.3 Our Contributions We design a small space streaming algorithm, MG-Triangle, to estimate transitivity and triangle counts for multiple time windows on multigraphs. As mentioned earlier, the main technical contribution is in handling repeated edges without a separate storage-intensive deduplication process. We consider this work as a first step towards small space streaming analytics for real-world graph streams.

• **The multiedge problem:** We applied previous streaming triangle algorithms [20, 12, 2] on multigraph streams, and showed that they fail to give correct answers. Fig. 1a shows how all these algorithms converge as their storage increases to an incorrect estimate on a DBLP multigraph stream. Of course, these algorithms were designed with the assumption of simple graph streams, and have excellent convergence properties (Fig. 1b). These results show how repeated edges are a problem and why we need new algorithms for multigraph streams.

• **Theoretical and empirical proofs of convergence:** We give proofs of convergence for MG-Triangle. Our algorithm is based on wedge sampling [23, 25] and borrows ideas from [12, 2]. It is provably correct on expectation. We also prove variance bounds, but MG-Triangle shows much better performance in practice than such bounds would indicate. We perform detailed experiments to prove that our algorithm gives accurate estimates with little storage (less than 5% of the stream in all instances). In Fig. 1a, we observe how MG-Triangle converges to the correct value storing at most 60K edges (the stream size is 3M).

• **Low storage required on real-world graphs:** Our algorithm stores less than 5% of the stream in all instances, and gives accurate estimates for transitivity and triangles counts. For example, we converted a 223M edge orkut graph [27] to a 500M edge multigraph, where our algorithm produced triangles estimates within 1% relative error. The storage required was just 1.2M edges, less than 0.5% of the stream. Our algorithm's worst performance (on a livejournal social network) only led to 0.04 additive error in transitivity, and 8.7% relative error in triangle count.

• **Multiple time window estimates in real-world graph streams:** Fig. 2 presents an example output of MG-Triangle on a DBLP coauthorship graph stream. MG-Triangle makes a single pass and stores less than 100K edges ($<$ 3% of total stream). It gives estimates for transitivity and triangles count at every year for window sizes of 5, 10, 15, 20 years, and all of time. In other words, at year (say) 2013, it gives triangle estimates for the simple graphs that aggregates edges in the following intervals: 2009–2013, 2004–2013, 1999–2013, 1994–2013, and 1938–2013. We immediately detect specific trends for different windows, like increasing window size decreases transitivity (even though triangle count naturally goes up). Also note the overall decrease of transitivity over time. We also

perform such analyses on an email network and a social network, and observe differences between these graphs.

2 Effects of repeated edges on triangle counting

We describe previous practical streaming triangle algorithms and explain why repeated edges is a challenge. We hope that this provides better context for our work and explains how important the assumption of simple graphs is for previous work. Our focus is on the neighborhood sampler of Pavan et al. [20], the wedge sampler of Jha et al. [12], and the sample-and-hold algorithm of Ahmed et al. [2]. To the best of our knowledge, these are the algorithms with established practical performance and good theoretical guarantees. (We omit the algorithm of Buriol et al. [5], since its practical performance is not good even for million edge streams [12].) For the sake of exposition, we formulate and describe the algorithms in slightly different terms from the original papers.

Reservoir sampling vs hashing: All algorithms sample uniform random edges from the stream, either by reservoir sampling or sampling an edge with fixed probability, which poses a problem in multigraph streams, since frequent edges have a higher probability of being sampled. This problem can be mitigated by using random hash functions. Suppose we wish to store each edge of the underlying simple graph from the stream with probability α. Each edge should be equally likely to be selected, independent of its frequency. Let *hash* be a uniform random function into the range $(0, 1)$. When the algorithm sees an edge e in the stream, it stores the edge if $hash(e) < \alpha$. Observe that the probability that an edge is selected only depends on its hash value and *is independent of its frequency.* We also stress that, for simple graph streams, hash based sampling is essentially equivalent to any other uniform random method.

Hashing provides an easy fix for the basic sampling problem, and is actually a convenient implementation method even for simple graphs. (We implemented all previous algorithm using hashing.) But the real challenge is debiasing, which comes next.

Neighborhood sampling [20]: Let edge f be a neighbor of e, if e and f intersect. The main idea of [20] is to pick a uniform random edge e, and then pick a uniform random neighbor f of e from the subsequent edges. This provides a wedge $\{e, f\}$, which is then checked for closure to provide a triangle. This process samples triangles non-uniformly. Pavan et al. cleverly debias by counting the number of following edges adjacent to e. (Equivalently, keeping track of the degree of vertices after storing e.) The algorithm takes a number of independent samples to get a low-error estimate. The method is provably correct and has excellent behavior in practice.

But multigraphs affect this debiasing. Tracking (simple) degrees of a vertex v is a non-trivial task, and requires counting the number of distinct edges incident to v. This itself requires a space overhead and it is not clear how to get a complete small-space extension of this approach for multigraphs.

Sample-and-hold [2] and wedge sampling [12]: Ahmed et al. give an elegant algorithm for triangle counting. Simply store every edge with some fixed (small) probability. For every edge e in the stream, count the number of triangles formed by e and a wedge among the stored edges. The sum of these counts can be used to estimate the total number of triangles. The final algorithm is simple, converges extremely rapidly, and is space efficient (To date, it is arguably the best streaming triangle counting algorithm). The wedge sampling algorithm of Jha et al [12] can also be thought of in this framework, except that it tracks a subset of the wedges created by stored edges.

Without getting into details, it suffices to say that the correctness of these algorithms hinges on a critical fact. Every triangle (in a simple graph) stream has a unique wedge that closes in the future. Suppose edges $\{e, f, g\}$ form a triangle, and edges appear in order $e, \ldots, f, \ldots, g$. Then the wedge $\{e, f\}$ is closed subsequently by edge g. It can be shown that both algorithms sample triangles uniformly, leading to unbiased estimates. This is not true for multigraph streams. If they appear in the stream as $e, f, g, e, f, g, e, f, g, \ldots$, there is no unique wedge closed in the future. (Indeed, all wedges are closed in the future.) This is a significant problem and increasing storage does not mitigate this problem. As demonstrated in Fig. 1a, these algorithms converge to an incorrect estimate as storage increases.

3 Proposed algorithm

Our algorithm MG-Triangle takes as input sampling rates $\alpha, \beta \in (0, 1)$ and a window Δt. The window is specified as a fixed number of edges or a fixed interval of time (like month, year, etc.). We describe the data structures used by MG-Triangle.

- Lists *e-list*, *w-list*: These are lists consisting of random edges and wedges, respectively. The sizes of these lists are controlled by α and β.
- Flags X_w: For each wedge $w \in$ *w-list*, we have a boolean flag X_w supposed to denote whether it is open or closed.

As mentioned earlier, it is convenient to think of *hash* as a uniform random function into the range $(0, 1)$.

Abusing notation, we will use *hash* to map various different objects[1] such as edges, wedges, etc.

Algorithm 1: MG-TRIANGLE$(\alpha, \beta, \Delta t)$

foreach *edge* e_t *in the stream* **do**
  Call `update`(e_t).
  foreach *wedge* w *in w-list* **do**
    Let $w = \{(u, v), (u, w)\}$.
    if e_t *is the closing edge* (v, w) **then**
      Set X_w to 1.
    else if $e_t \in \{(u, v), (u, w)\}$ **then**
      Reset X_w to 0. `// bias-correction`
  Let $\mathcal{W} \subseteq$ *w-list* be the set of wedges that formed in time $[t - \Delta t, t]$.
  Output $\widehat{T}_t = (\alpha^2\beta)^{-1} \sum_{w \in \mathcal{W}} X_w$.
  Output $\widehat{W}_t = (\alpha^2\beta)^{-1}|\mathcal{W}|$ and $\tau_t = 3\widehat{T}_t/\widehat{W}_t$ (if $\widehat{W}_t = 0$, set $\tau_t = 0$).

Algorithm 2: `update`(e_t)

if $hash(e_t) \leq \alpha$ *and* $e_t \notin$ *e-list* **then**
  Insert e_t in *e-list*.
  foreach *wedge* $w = (e, e_t)$ *where* $e \in$ *e-list* **do**
    if $hash(w) \leq \beta$ *and* $w \notin$ *w-list* **then**
      Insert w in *w-list*.

3.1 High level description The first step on encountering edge e_t is to update the lists *e-list* and *w-list*. This is done in procedure `update`. The idea is based on standard hash-based sampling. We add e_t to *e-list* if $hash(e_t) \leq \alpha$ and e_t is not already in *e-list*. Then, we look at all the wedges that e_t creates with existing edges in *e-list*. We apply another round of hash-based sampling to put these wedges in *w-list*.

Critically, if an edge e enters *e-list*, it never leaves. If e enters *e-list*, it does so the first time it appears in the stream. The probability of an edge entering *e-list* is independent of its frequency in the stream. This is vital to get unbiased samples of edges in the underlying simple graph G_t. Similar statements hold for wedges.

Checking for closures and debiasing: We encounter edge e_t and have updated *e-list* and *w-list*. For each wedge $w \in$ *w-list*, we have a boolean variable X_w. If e_t closes w (so w and e_t form a triangle), we set $X_w = 1$. This is the standard wedge-sampling approach [23, 25, 12]. At this point, the algorithm would basically be that of [12], implemented with hash-based sampling. As argued earlier and shown in Fig. 1, this algorithm does not work.

To fix the biasing, we perform a somewhat mysterious step. We have wedge $w \in$ *w-list* and encounter e_t. If e_t is already part of w, we simply reset X_w to 0. So even though w may be closed, we just assume it is open. This completely resolves the biasing, and we give a formal proof in Thm. 3.3.

Outputting the estimate: Finally, we need to output estimates,$|\widehat{T}_t|, |\widehat{W}_t|, \widehat{\tau}_t$ for $|T_t|, |W_t|, \tau_t$, respectively. This is the only step where the time window Δt is used. We look at all wedges in *w-list* that formed in the time $[t - \Delta t, t]$. The total number of these wedges can be scaled to estimate $|W_t|$. The number of these wedges, where $X_w = 1$ is scaled to estimate $|T_t|$, and the appropriate ratio estimates τ_t.

3.2 Theoretical analysis We prove that the MG-TRIANGLE is correct on expectation and prove weak concentration results bounding the variance. We also show some basic bounds on the storage of MG-TRIANGLE. Throughout this section, we focus at some time t and the simple graph G_t. We stress that there is no distributional assumption on the graph or the stream. All the probabilities are over the internal randomness of the algorithm (which is encapsulated in the random behavior of *hash*).

LEMMA 3.1. *Consider time t. For any edge $e \in G_t$, the probability that $e \in$ e-list is α. For any wedge $w \in W_t$, the probability that $w \in$ w-list is $\alpha^2\beta$.*

Proof. Consider edge e. We first argue that $e \in$ *e-list* iff $hash(e) \leq \alpha$ (Note that this is independent of the frequency of e). Suppose $hash(e) \leq \alpha$. At its first occurrence, e enters *e-list* and remains in *e-list*. Suppose $hash(e) > \alpha$. At no timestep will e be added to *e-list*, regardless of how many times it appears. From the randomness of *hash*, $hash(e) \leq \alpha$ with probability α. Hence, $e \in$ *e-list* with probability α.

For wedge $w = \{e, e'\}$ to be in *w-list*, both its edges must be in *e-list*. That means both $hash(e)$ and $hash(e')$ are at most α. Suppose the first occurrence of e is before that of e'. At the first time e' occurs, procedure `update` will add w to *w-list* iff $hash(w) \leq \beta$. At any subsequent occurrence of e or e', the wedge w is not considered for adding to *w-list* (simply because e and e' are already in *e-list*). The total probability (by the randomness of *hash*) is $\alpha^2\beta$. □

The following hold just by linearity of expectation. We move proofs to the supplementary file attached with the submission.

[1] This is implemented by appropriately concatenating vertex ids.

THEOREM 3.1. *The expected size of e-list is* $\alpha E(G[1,t])$ *and the expected size of w-list is* $\alpha^2\beta W(G[1,t])$.

THEOREM 3.2. $\mathbf{E}[\widehat{W}_t] = |W_t|$.

Now we come to a key theorem that shows that $\widehat{T}_t$ is correct on expectation. This is where we prove that our proposed debiasing technique works.

THEOREM 3.3. $\mathbf{E}[\widehat{T}_t] = |T_t|$.

Proof. We extend the definition of Boolean flag X_w to every wedge w in W_t. Let $X_w = 0$ if w is not present in *w-list* (at time t). Note that $\widehat{T}_t = (\alpha^2\beta)^{-1}\sum_{w\in W_t} X_w$. For every edge e in E_t, let $t_{max}(e)$ be the maximum time $s \le t$ such that $e_s = e$. Fix a triangle $A = \{a,b,c\} \in T_t$ formed by edges a, b, and c, and assume (by relabeling if required) that c is the last edge to appear in the stream among a, b, and c. In other words, $t_{max}(c) > \max\{t_{max}(a), t_{max}(b)\}$. Since $\{a,b\},\{b,c\},\{c,a\}$ are wedges, it makes sense to talk about $X_{\{a,b\}}$, etc. The following is the debiasing argument, showing that exactly one wedge in A has $X_w = 1$.

LEMMA 3.1. $X_{\{b,c\}} = X_{\{c,a\}} = 0$. *Moreover,* $X_{\{a,b\}} = 1$ *iff* $\{a,b\}$ *is in w-list.*

Proof. Consider the moment $s = t_{max}(c)$ when $e_s = c$. If wedge $\{b,c\} \notin$ *w-list*, then by definition, $X_{\{b,c\}}$ is 0. If $\{b,c\} \in$ *w-list*, then by Step 8 of Algorithm 1, the value of $X_{\{b,c\}}$ is reset to 0. No subsequent change is made to this value. An identical argument shows the same for $X_{\{c,a\}}$. Finally, $X_{\{a,b\}}$ is set to 1 at this moment iff if wedge $\{a,b\}$ is in *e-list*, and once again, this value is not changed subsequently. □

By Lemma 3.1, $\mathbf{E}[X_{\{b,c\}}] = \mathbf{E}[X_{\{c,a\}}] = 0$, while $\mathbf{E}[X_{\{a,b\}}]$ is the probability that this wedge is in *w-list*. This is exactly $\alpha^2\beta$. Therefore, the sum of expectations of X_w over all three wedges w of the triangle $A = \{a,b,c\}$ is $\sum_{w\in A}\mathbf{E}[X_w] = \alpha^2\beta$. Observe this is true for any fixed triangle in T_t. For any wedge w that does not participate in a triangle, X_w is obviously zero. By linearity of expectation, $\mathbf{E}[\widehat{T}_t] = (\alpha^2\beta)^{-1}\mathbf{E}[\sum_{w\in W_t} X_w] = (\alpha^2\beta)^{-1}\sum_{A\in T_t}\sum_{w\in A}\mathbf{E}[X_w]$. Plugging in the value of $\mathbf{E}[X_w]$, this is $(\alpha^2\beta)^{-1}\cdot\alpha^2\beta|T_t| = |T_t|$. □

Using methods from [12], we can prove weak concentration bounds for $\widehat{T}_t$ and $\widehat{W}_t$ (by bounding their variance). We need to assume that α and β are large enough to ensure that enough wedges of W_t are in *w-list*, and there are at least as many wedges in G_t as edges. The latter is needed to rule out extreme cases like G_t being a path or a matching. This assumption is reasonable for real-world networks, as can be seen in Tab. 1. Proof is in the supplementary file attached with the submission.

THEOREM 3.4. *Fix some sufficiently small* $\gamma > 0$. *Suppose that* $(\alpha^2\beta)|W_t|$ *(the expected number of wedges in* W_t *that are in w-list) is at least* $1/\gamma^6$. *Furthermore* $|W_t| \ge |E_t|$ *(there are at least as many wedges in* G_t *as edges). Then,* $\Pr[|\widehat{W}_t - |W_t|| > \gamma|W_t|] < \gamma$, $\Pr[|\widehat{T}_t - |T_t|| > \gamma|W_t|] < \gamma$, *and* $Pr[|\widehat{\tau}_t - \tau_t| > 8\gamma] < 4\gamma$.

4 Empirical evaluation of MG-TRIANGLE

We implemented our algorithm in C++ and ran it on a MacBook Air laptop with 1.7 GHz Intel Core i7 processor and 8 GB 1600 MHz DDR3 RAM. We applied MG-TRIANGLE on a variety of real-world datasets. Refer to Tab. 1 for details about these datasets.

`DBLP`**:** This is a co-authorship network for papers on the DBLP website. From the raw data at DBLP [15] we extracted 786,719 papers by ignoring papers with (i) a single author, (ii) more than 100 authors, and (iii) missing "year" metadata. For each paper we put an edge corresponding to every distinct pair of co-authors resulting in a total of 3,630,374 (multi)edges.

`Enron`**:** This network is derived from emails between Enron employees between 1999 and 2003 [22]. Nodes correspond to employees while edges represent their email correspondence. Multiple emails between the same pair of individuals result in a multigraph.

`Flickr`**:** This dataset consists of friendship connections of users of Flickr, obtained from [22]. Originally, the data was collected in [19]. (Results on `Flickr` given in the supplementary file attached with the submission.)

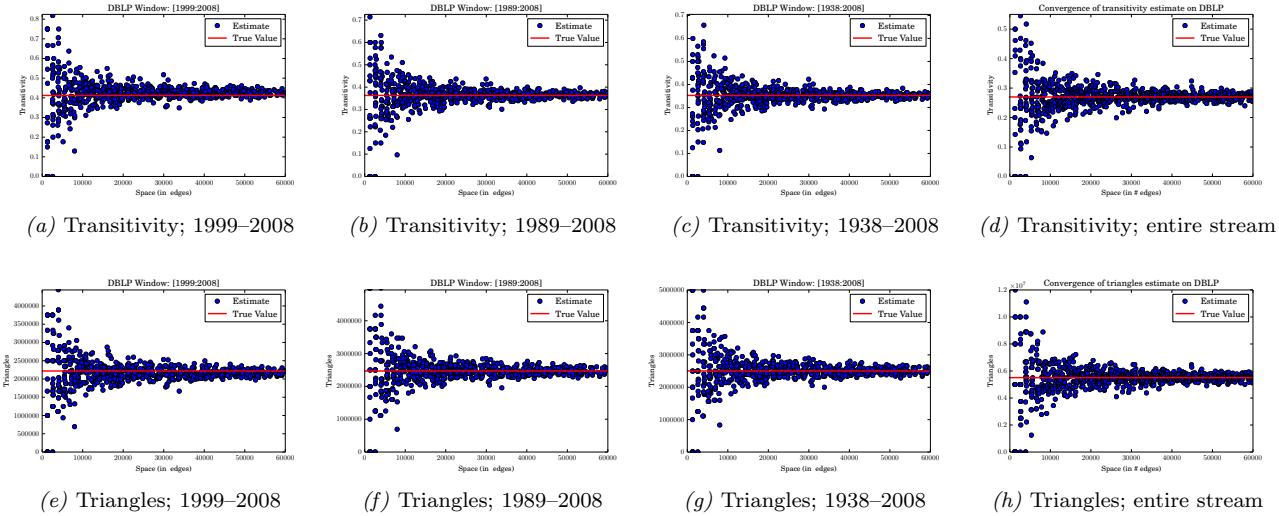
SNAP: We extended our data set to include networks from SNAP [27]. We synthetically replicate edges of these datasets to get a multigraph.

Convergence of estimate: Fig. 3(d) and Fig. 3(h) demonstrate convergence of the *final estimates* (i.e. for G_m) for increasing space. We define storage as the number of edges stored by our algorithm: $|e\text{-}list| + 2\cdot|w\text{-}list|$. We first choose β in $\{0.2, 0.4, 0.6, 0.8, 1.0\}$ and then vary α in increments of 0.0005 up to 0.02. For each setting of α and β, we plot 5 runs of the algorithm. One can see that both the transitivity and triangles estimates converge rapidly to true values as we increase the space.

Our estimates for various time windows also converge rapidly, as we demonstrate in Fig. 3. For these experiments, we picked specific time windows on `DBLP`, namely, 1989–2008, 1999–2008, and 1938–2008. This is mostly for demonstrating the convergence of differing window sizes. We chose β from $\{0.2, 0.4, 0.6, 0.8, 1.0\}$ and varied α in increments of 0.1% up to 3.0%. For each value of α and β, we give 5 runs of the algorithm.

Table 1: A run of our algorithm on a variety of real-world and synthetic graphs with $\alpha = 0.01$ and β set such that size of *w-list* is at most 50K. The third column gives the number of edges in the multigraph while the fourth column (Space) gives the space (in terms of number of edges) used by the algorithm. The first three datasets are raw real-world datasets whereas the remaining datasets were synthetically made multigraphs starting with graphs from [27].

Dataset	n	Wedges	Edges		Space	Transitivity		Triangles	
		(simple)	simple	multi		exact	estimate	exact	Rel. error
DBLP	755K	61M	2.54M	3.63M	31K	0.269	0.282	5.50M	3.09%
Enron	86K	49M	297K	1.15M	8K	0.069	0.071	1.18M	3.38%
Flickr	2302K	22B	22M	33.1M	251K	0.110	0.108	837M	1.24%
as-skitter	1.6M	16B	11M	53M	160K	0.005	0.005	28M	6.50%
cit-Patents	3.7M	0.3B	16M	79M	199K	0.067	0.066	7.51M	0.33%
web-Google	0.8M	0.7B	4M	20M	79K	0.055	0.057	13.3M	3.79%
web-NotreDame	0.3M	0.3B	1M	5M	42K	0.088	0.088	8.91M	3.93%
youtube	1.1M	1.4B	2M	14M	64K	0.006	0.006	3.05M	1.86%
livejournal	5.2M	7.5B	48M	205M	473K	0.124	0.118	310M	8.65%
orkut	3.0M	45B	223M	562M	1.2M	0.041	0.041	627M	0.09%



(a) Transitivity; 1999–2008 *(b)* Transitivity; 1989–2008 *(c)* Transitivity; 1938–2008 *(d)* Transitivity; entire stream

(e) Triangles; 1999–2008 *(f)* Triangles; 1989–2008 *(g)* Triangles; 1938–2008 *(h)* Triangles; entire stream

Figure 3: DBLP convergence: We show that both transitivity and triangles estimates converge to true values as we increase the space. The top row is for transitivity, while the bottom row is for triangle counts. The plots are arranged in the order of increasing window length ending at 2008 except for the last column which corresponds to the entire stream length.

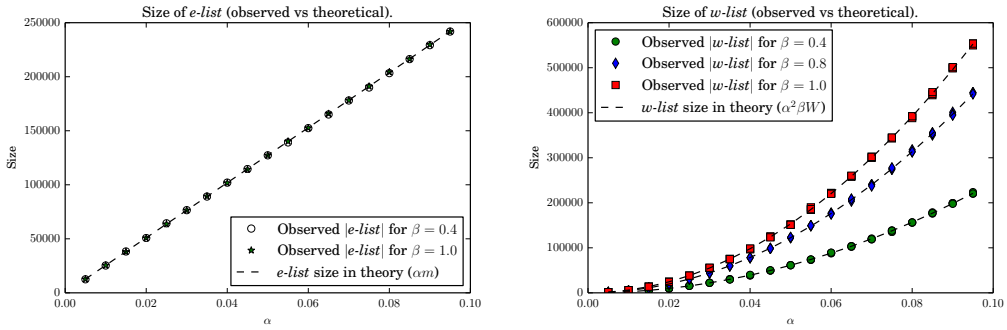
In the plots x-axis gives increasing space (i.e., increasing α) and the y-axis is the estimate.

Across the board, we see rapid convergence as storage increases. For DBLP, storage of 60K is enough to guarantee extremely accurate results (relative errors within 5%), for all the time windows. This is even true for the 10 year window, which is quite small compared to the entire stream of data (MG-Triangle will not work for window sizes of a year, since there are not enough samples from such a window. But the number of edges in a year is small enough to store explicitly).

Space usage: Fig. 4 shows the space used by our algorithm in terms of parameters α and β. We measure both *e-list* and *w-list* for varying values of α and β, and plot the predictions of Thm. 3.1. We see almost perfect alignment of the predictions with Thm. 3.1.

Comparison with previous work: We run the algorithms of [20], [12], and [2], using hash based sampling to recreate uniform edge sampling in a multigraph. We first note that our implementations work correctly on the simple graph version of DBLP, shown in Fig. 1b. All algorithms converge extremely rapidly. When these algorithms are applied to the multigraph version of DBLP, then they all converge to incorrect triangle estimates (Fig. 1a).

Tests on a broader data set: For more validation of MG-Triangle, we run it on a large set of real-world graphs. Most of these graphs are neither temporal nor multigraphs. We construct a multigraph stream from each graph as follows: every edge e of the graph is independently replicated with probability 1/3 (specifically r times where r is uniform in $\{2, 4, 8, 16, 32\}$). The stream is obtained by randomly permuting these multiedges. For each graph, we only use MG-Triangle



(a) e-list size *(b) w-list* size

Figure 4: Number of edges and wedges stored by *e-list* and *w-list* respectively. The observed usage matches almost exactly with theoretical predictions.

record to transitivity and triangle count of the entire stream (the graph $G[1, m]$). The results are presented in Tab. 1. For these runs, we set $\alpha = 0.01$ and capped the size of wedge reservoir to $50K$ (by choosing β appropriately). We observe that transitivity estimates are very sharp (matching the true values up to the third decimal point in many cases). The relative error in triangles estimates is less than 3% for most cases and never exceeds 8.7%. The overall space used by the algorithm is at most 4% of the number of edges of the underlying *simple* graph. We point out that for `orkut` which has nearly half a billion edges (after injecting duplicate edges), the transitivity estimate closely matches with the true value and the relative error in triangles is less than 1%. The total storage used is less than 0.5% of the edge stream.

5 Experiments with time windows

MG-Triangle takes as input a single time window length Δt. But observe that the primary data structures *e-list*, *w-list*, and X_w are independent of this window. As a result, MG-Triangle can handle multiple time windows with the *same* data structure. We only maintain the latest timestamp for each edge, and do not store any history. If the time window $[t - \Delta t, t]$ is too small, it is unlikely that *e-list* will have any edges from this window. On the other hand, small time windows can be stored explicitly to get exact answers.

Triangle trends in `DBLP`: In our opinion, the following results are the real achievement of MG-Triangle. We wish to understand transitivity and triangle trends for `DBLP` in various time windows. We focus on 5-year, 10-year, 15-year, 20-year, and entire history windows. So think of a (say) 5-year sliding time window in `DBLP`, and the aim is to report the transitivity in each such window. Refer to Fig. 2 ("All" refers to the window that contains the entire history). *The algorithm* MG-Triangle *makes a single pass over* `DBLP` *without preprocessing and provides results for all these windows at every year.*

The transitivity reveals intriguing trends. Firstly, smaller windows have higher transitivity. It shows that network clustering tends to happen in shorter time intervals. This is probably because of the affiliation structure of coauthorship networks. The increase of triangle counts over time (for the same window size) may not be too surprising, given that the volume of research increasing. But juxtapose this with the *decreasing* of transitivity over time. This means that (say) the transitivity in 2004–2008 is higher than 2009–2013, even though there are more papers (and more triangles) in the latter interval. Why is this the case? Is it because of increasing of interdisciplinary work, which might create more open wedges? Or is it simply some issue with the recording of `DBLP` data? Will the decreasing transitivity converge in the future, or do we expect it to simply go to zero? Can we give a reasonable model of this behavior? We believe that the output of MG-Triangle will lead to many data science questions, and this is the real significance of the algorithm.

Triangle trends in `Enron`: In Fig. 2d and Fig. 2c, we present triangles and transitivity estimates for `Enron` for various windows. For this dataset, we think of a window as being defined by a specified number of past edges. In particular, apart from considering the entire past, we look at windows formed by past 200K, 400K, and 800K edges. Observe that in the beginning of the stream all these windows coincide, since the windows are equivalent. Focusing on the triangles estimate, it is clear that the estimate corresponding to the larger window size dominates that of a smaller window size. What is

interesting for Enron dataset is that the same ordering is observed even for transitivity estimates. That is, in general, a transitivity estimate curve corresponding to the larger size window dominates the one corresponding to the smaller size. We observe a completely opposite behavior with DBLP transitivity curves, see Fig. 2.

Another interesting observation is that in case of Enron, the curves for triangles estimates for smaller window lengths flattens out whereas that in DBLP the curves for triangle estimates continue to rise even for smaller time windows. This indicates that the growth of *total number of triangles* is superlinear in DBLP (with respect to the number of years) whereas it is nearly linear (with respect to the number of edges seen so far) in case of Enron. Indeed the final estimate for the number of triangles in Enron is almost the same as the number of edges in the stream.

Acknowledgements

We thank Ashish Goel for suggesting the use of hash-function based reservoir sampling. This was a key step towards the development of the final algorithm.

References

[1] N. Ahmed, J. Neville, and R. Kompelle. Network sampling: From static to streaming graphs. *ACM TKDD*, 8(2):7:1–7:56, 2013.

[2] N. K. Ahmed, N. Duffield, J. Neville, and R. Kompella. Graph sample and hold: A framework for big graph analytics. In *KDD*, 2014.

[3] K. J. Ahn, S. Guha, and A. McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *PODS*, pages 5–14, 2012.

[4] L. Becchetti, P. Boldi, C. Castillo, and A. Gionis. Efficient semi-streaming algorithms for local triangle counting in massive graphs. In *KDD*, pages 16–24, 2008.

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

[6] R. S. Burt. Structural holes and good ideas. *American J. Sociology*, 110(2):349–399, 2004.

[7] J. S. Coleman. Social capital in the creation of human capital. *American J. Sociology*, 94:S95–S120, 1988.

[8] G. Cormode and S. Muthukrishnan. Space efficient mining of multigraph streams. In C. Li, editor, *PODS*, pages 271–282. ACM, 2005.

[9] N. Durak, A. Pinar, T. G. Kolda, and C. Seshadhri. Degree relations of triangles in real-world networks and graph models. In *CIKM'12*, 2012.

[10] 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.

[11] D. F. Gleich and C. Seshadhri. Vertex neighborhoods, low conductance cuts, and good seeds for local community methods. In *KDD*, 2012.

[12] M. Jha, C. Seshadhri, and A. Pinar. A space efficient streaming algorithm for triangle counting using the birthday paradox. In *KDD*, KDD '13, pages 589–597, New York, NY, USA, 2013. ACM.

[13] H. Jowhari and M. Ghodsi. New streaming algorithms for counting triangles in graphs. In *COCOON*, pages 710–716, 2005.

[14] D. M. Kane, K. Mehlhorn, T. Sauerwald, and H. Sun. Counting arbitrary subgraphs in data streams. In *ICALP*, pages 598–609, 2012.

[15] M. Ley. Digital bibliography & library project. http://www.informatik.uni-trier.de/~ley/db/.

[16] S. Macskassy. Mining dynamic networks: The importance of pre-processing on downstream analytics. In *P. Intl W. Mining Communities and People Recommenders*, 2012.

[17] A. McGregor. Graph stream algorithms: a survey. *ACM SIGMOD Record*, 43:9–20, 2014.

[18] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. Network motifs: Simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002.

[19] A. Mislove, H. S. Koppula, K. P. Gummadi, P. Druschel, and B. Bhattacharjee. Growth of the flickr social network. In *ACM SIGCOMM W. Social Networks*, August 2008.

[20] A. Pavan, K. Tangwongsan, S. Tirthapura, and K.-L. Wu. Counting and sampling triangles from a graph stream. In *VLDB*, 2013.

[21] A. Portes. Social capital: Its origins and applications in modern sociology. *Annual Rev. Sociology*, 24(1):1–24, 1998.

[22] R. Rossi and N. Ahmed. Network repository, 2013.

[23] T. Schank and D. Wagner. Approximating clustering coefficient and transitivity. *J. Graph Algorithms and Applications*, 9:265–275, 2005.

[24] C. Seshadhri, T. G. Kolda, and A. Pinar. Community structure and scale-free collections of Erdös-Rényi graphs. *Physical Review E*, 85(5):056109, May 2012.

[25] C. Seshadhri, A. Pinar, and T. G. Kolda. Triadic measures on graphs: The power of wedge sampling. In *SDM*, 2013.

[26] C. Seshadhri, A. Pinar, and T. G. Kolda. Wedge sampling for computing clustering coefficients and triangle counts on large graphs? *Statistical Analysis and Data Mining*, 7(4):294–307, 2014.

[27] SNAP. Stanford network analysis project, 2013. Available at http://snap.stanford.edu/.

[28] K. Tangwongsan, A. Pavan, and S. Tirthapura. Parallel triangle counting in massive streaming graphs. In *CIKM*, 2013.

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

[30] B. F. Welles, A. V. Devender, and N. Contractor. Is a friend a friend?: Investigating the structure of friendship networks in virtual worlds. In *CHI-EA'10*, pages 4027–4032, 2010.