SAND2014-15833C

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

Madhav Jha
Sandia National Laboratories
7011 East Avenue
Livermore, CA 94550
Email: mjha@sandia.gov

Ali Pinar
Sandia National Laboratories
7011 East Avenue
Livermore, CA 94550
Email: apinar@sandia.gov

C. Seshadhri
Sandia National Laboratories
7011 East Avenue
Livermore, CA 94550
Email: scomand@sandia.gov

***Abstract*—Graphs in the real-world are often temporal and can be represented as a "stream" of edges. Estimating the number of triangles in a graph observed as a stream of edges is a fundamental problem in data mining. Our goal is to design a single pass space-efficient streaming algorithm for estimating triangle counts. While there are numerous algorithms for this problem, they all (implicitly or explicitly) assume that the stream does not contain duplicate edges. However, real graph streams are rife with duplicate edges. The work around is typically an extra unaccounted pass (storing all the edges!) just to "clean up" the data. Furthermore, previous work tends to aggregate *all edges* to construct a graph, discarding the temporal information. It will be much more informative to investigate temporal windows, especially multiple time windows simultaneously.**

Can we estimate triangle counts for multiple time windows in a single pass even when the stream contains repeated edges? In this work, we give the first algorithm for estimating the triangle count of a multigraph stream of edges over arbitrary time windows. We build on existing "wedge sampling" work for triangle counting. Duplicate edges create significant biasing issues for small space streaming algorithms, which we provably resolve through a subtle debiasing mechanism. Moreover, our algorithm seamlessly handles multiple time windows. The final result is theoretically provable and has excellent performance in practice. Our algorithm discovers fascinating transitivity and triangle trends in real-world temporal graphs.

I. Introduction

The abundance of triangles has been observed in networks arising from numerous scenarios, such as social interaction, coauthorship, citations, communications, etc. This abundance is noted as a critical feature that distinguishes real graphs from random graphs. In social sciences, triangle counts are used as a guide to understand graphs [1], [2], [3], [4]. Triangle counts are also used in some graph mining applications such as spam detection [5], community detection [6], [7] and finding common topics on the WWW [8]. Frequency of triadic patterns is a standard part of motif detection in bioinformatics [9]. Triangles are also used in modeling and characterizing real-world networks [10], [11].

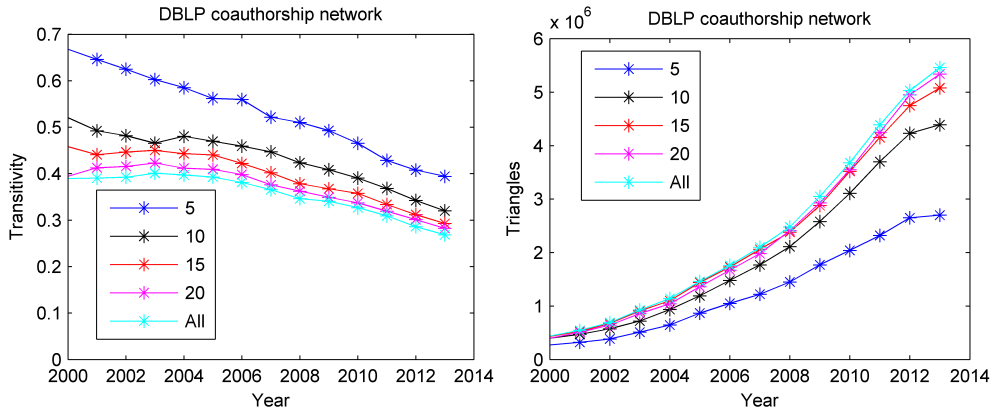
Many massive graphs are truly temporal and manifest in practice as a *stream of edges*. People call each other on the phone, exchange emails, or co-author a paper; computers exchange messages; animals come in the vicinity of each other; companies trade with each other. Each such interaction is modeled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

Due to the sheer 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 we may have, for example, $e_1 = e_{100} = e_{125} = (u, v)$. We are interested in *small space streaming algorithms* that make a *single pass* over the stream $e_1, e_2, \ldots, e_m$. At any timestep t, such an algorithm retains a very small (possibly random) subset of the edges seen so far. This is called the "sketch" and is updated rapidly as new edges appear. Using the sketch and some auxiliary data structures, the algorithm computes an accurate estimate for the number of triangles for the graph seen so far. The size of the data structures is orders of magnitude smaller than the size of the graph. 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 already appeared before.

There is much work on streaming graph algorithms [12], [13], [14], [15], [16] (see theoretical survey [17]). Yet all of the practical algorithmic work ignores important issues such as repeated edges and temporal aggregation that arise when looking at a real-world graph stream.

Graph vs multigraph: Previous results assume that the edge stream forms a *simple graph*, and no edge is ever repeated in the stream. Indeed, a recent survey on network sampling explicitly defines a graph stream as a permutation of the edges of the underlying simple graph [14]. This is a useful assumption for algorithmic progress; on the other hand, it is almost never true in practice. Real-world graph streams are truly *multigraphs*, in that same edges can repeatedly occur 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 [18], [19]) are repeated.

The assumption of simplicity is actually 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.* Our aim is to count the triangles in the underlying simple graph in a single pass over the multigraph stream. There is no existing result that can do this. We posit that for streaming algorithms to be actually



(a) Transitivity (b) Triangles

Fig. 1: Transitivity and triangles estimates for varying window lengths for DBLP coauthorship graph: Our algorithm stores less than 3% of the multigraph stream yet produces fine-grained triangle results.

useful in practice, the aspect of multiple edges must be dealt with.

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 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 the sake of this paper, let us 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 by an artifact of the window size [20]. Therefore, it is essential to observe multiple time windows at the same time, instead of committing to single one.

A. Triangle counting

Our focus is on the classic problem of triangle counting in graph streams, which has received much attention in recent years [12], [13], [14], [15], [16]. We stress that none of these results deal with multigraphs or time aggregation.

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 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.

Challenges: Multigraphs are a major headache for streaming algorithms, especially for triangle counting. 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 $a, b, c, a, b, c, a, b, c, \ldots$. (Observe how this is *not* an issue for simple graphs.) These patterns create biases for existing triangle counting algorithm, which we explain in more detail later. For now, it suffices to say that existing algorithms [12], [13], [14], [15], [16] will give different estimates for triangle counts of different multigraphs streams that contain the same simple graph. Previous work on multigraph mining explicitly states triangle counting of streaming multigraphs as an open problem [21].

Maintaining *unbiased* estimates for numerous time windows is tricky. State-of-the-art triangle counting algorithms keep degrees of vertices to appropriately set sampling probabilities [15], [16] or use fairly involved wedge replacement methods [13]. It is not clear whether such information can be maintained for multiple time windows without significantly increasing storage. Compounded with multiple edges, where the same edge can have different frequencies for different time windows, previous algorithms are only able to report results for the entire history of edges.

B. Our Contributions

We design a small space streaming algorithm that estimates transitivity and triangle counts for multiple time windows on multigraphs. This is the first streaming algorithm for triangle counting to handle either of these aspects. We consider this work as a first step towards small space streaming analytics for real-world graph streams.

• **Multiple time window estimates in real-world graph stream:** An example output of our algorithm is given in Fig. 1. We consider a DBLP coauthorship graph stream, where each edge represents two individuals writing a paper together. The graph stream has over 3 million multiedges, and 2 million simple edges. Our algorithm makes a single pass and stores less than 100K edges ($< 3\%$ of total stream). It gives estimates for the 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 obtained by aggregating edges in the following

intervals: [2008, 2013], [2003, 2013], [1998, 2013], [1993, 2013], and [1900, 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.

• **Theoretical and empirical proofs of convergence:** We give mathematical proofs of convergence for our algorithm. Our randomized algorithm is inspired by previous work on wedge sampling [22], [23] and borrows techniques from [13]. It is relatively simple and clean, and is provably correct on expectation. Leveraging previous work, we can show variance bounds, but the algorithm has much better performance in practice than such bounds would indicate. We perform detailed experiments on numerous datasets to prove that our algorithm gives accurate estimates with little storage (less than 5% of the stream in all instances).

• **Debiasing multiedges:** Even to give an estimate for the entire stream of edges (so no limited time windows), previous results did not work for multigraphs. We demonstrate this by showing that the algorithm [13] fails quite spectacularly on the DBLP stream. We require a subtle debiasing mechanism, and empirically (and theoretically) show it is necessary for accurate estimates.

• **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 estimates. For example, we converted a 223M edge `orkut` graph obtained from SNAP [24] to a *half a billion* edge multigraph to test our results. On this synthetic dataset, our algorithm produced triangles estimates accurate within 1% relative error and transitivity estimates matching the exact value up to three decimal places. The storage required was just 1.2M edges ($< 0.5\%$ 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.

C. Related Work

The most closely related work from the perspective of computing on multigraph streams are [25] and [21]. As mentioned earlier, [21] explicitly mentions the question of counting subgraphs in multigraph as directions for future work.

There is significant history on triangle counting in various settings. There are algorithmic methods to deal with such massive graphs, such as random sampling [22], [26], [23], MapReduce paradigm [27], [28], distributed-memory parallelism [29], [30], adopting external memory [31], [32], and multithreaded parallelism [33]. There is much work on triangle counting in graph streams [34], [12], [35], [36], [16], [15], [13]. *All of these methods start with preprocessing the data to remove duplicate edges, an expensive operation.* We also refer the reader to a recent tutorial on network sampling for an overview of random sampling techniques [37].

A reader may wonder why we ignore duplicates, and not consider them for a more involved triangle analysis. We agree that this is an interesting problem, and duplicates have their own significance. On the other hand, it is standard to focus on simple graphs for network analysis, and there is no consensus on how to define triangle counts, clustering coefficients, etc. on true multigraphs. We feel that this is an exciting avenue for future work.

D. 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 $\tau_t = \tau(G_t)$.

II. The Algorithm

Our main algorithm is `estimate`, which takes as input sampling rates $\alpha, \beta \in (0, 1)$ and a *time window* Δt. The window is usually specified as a fixed number of edges, a fixed interval of time (like month, year, etc.), though the algorithm works for windows lengths that change with time. We describe the important variables and data structures used by `estimate`.

• Lists *e-list*, *w-list*: These are random lists of edges and wedges, respectively. This is the bulk of the storage. 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 (but not quite) whether it is open or closed.

We will use a hash function, denoted $hash$. 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.

A. High level description and challenges

Maintaining lists by hash-based sampling: The first step on encountering edge e_t is to update the lists *e-list* and *w-list*. This is done in the 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*.

The critical aspect to note is that if an edge e enters *e-list*, then it never leaves. Furthermore, e will enter *e-list* the first time it appears in the stream. *So, the probability of an edge entering e-list is independent of its frequency in the stream.*

[1] This can easily be implemented by appropriately concatenating vertex ids.

Algorithm 1: `estimate`$(\alpha, \beta, \Delta t)$

foreach *edge* e_t *in the stream* **do**
  Call `update`(e_t).
  foreach *wedge* w *in e-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 $\widehat{W}_t = |\mathcal{W}|$ 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*.

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: This is the step where we actually look for triangles. 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 [22], [23], [13]. But this approach creates biases in multigraphs. We elaborate below.

Suppose there are three edges e, f, g that form a triangle and the stream had $e, e, e, \ldots, f, f, f, \ldots, g, g, g \ldots$. So each edge occurred in a disjoint stretch of time. The probability that `estimate` finds triangle $\{e, f, g\}$ is exactly the probability that wedge $\{e, f\}$ enters *w-list*. This is $\alpha^2\beta$ (Lem. 2.1). Thus, there is a specific wedge in $\{e, f, g\}$ which must be present in *w-list*.

Consider edges e', f', g' that form a triangle and the stream $e', f', g', e', f', g', \ldots$. So the stream cycles through these edges. The probability that `estimate` finds triangle $\{e', f', g'\}$ is *larger*. If *any* of the wedges $\{e', f'\}$, $\{f', g'\}$, or $\{e', g'\}$ enters *w-list*, the triangle is detected. This is because for all the wedges, the closing edge occurs in the future. The probability that this triangle is detected is roughly $3\alpha^2\beta$.

This is a major problem for triangle counting, since triangles do not have the same probability of detection. One may solve this probem by storing the entire histories of an edge, and trying to figure out this bias. *Surprisingly, we have a solution that requires no storage of history.* This is the critical debiasing step, which is deceptively simple.

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. 2.4.

Outputting the estimate: Finally, we need to output estimates (denoted by hatted variables) for $|T_t|, |W_t|, \tau_t$. 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 w where $X_w = 1$ is scaled to estimate $|T_t|$, and the appropriate ratio estimates τ_t.

B. Theoretical analysis

We prove that the `estimate` is correct on expectation, and prove some (weak) concentration results bounding the variance. We also show some basic bounds on the storage of `estimate`. A key (and simple) fact is the uniform sampling of edges and wedges. 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 2.1: 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. Its first occurrence is at some time $s \leq t$. At time s, e enters *e-list* iff $hash(e) \leq \alpha$. From the randomness of $hash$, this happens with probability α. For wedge w to be in *w-list*, both its constituent edges must be present in *e-list*. This happens with probability α^2. After this, it needs to be selected to go into *w-list*, which happens with probability β. ■

A simple application of linearity of expectation proves the storage bound (which depends on the entire graph, not just G_t).

Theorem 2.2: The expected storage of `estimate` at time t is $O(\alpha E(G[1, t]) + \alpha^2\beta W(G[1, t]))$.

Proof: The storage is dominated by the sizes of the lists *e-list* and *w-list*. For each edge e, let Z_e be the indicator for e being in *e-list* at time t. The expected size of *e-list* is $\mathbf{E}[\sum_{e \in E(G[1,t])} Z_e]$. By the previous claim $\mathbf{E}[Z_e] = \alpha$, and linearity of expectation completes the proof. An identical arguments holds for *w-list*. ■

A similar argument shows $\widehat{W}_t$ is correct on expectation.

Theorem 2.3: $\mathbf{E}[\widehat{W}_t] = |W_t|$.

Proof: For any wedge $w \in W_t$, let $Y_w = 1$ if $w \in$ *w-list* and 0 otherwise. Note that $\widehat{W}_t = (\alpha^2\beta)^{-1} \sum_{w \in W_t} Y_w$. We have $\mathbf{E}[Y_w] = \alpha^2\beta$ by Lem. 2.1. By linearity of expectation,

$$\begin{aligned} \mathbf{E}[\widehat{W}_t] &= (\alpha^2\beta)^{-1}\mathbf{E}[\sum_{w \in W_t} Y_w] \\ &= (\alpha^2\beta)^{-1} \sum_{w \in W_t} \mathbf{E}[Y_w] = |W_t| \end{aligned}$$

■

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

Theorem 2.4: $\mathbf{E}[\widehat{T}_t] = |T_t|$.

Proof: We extend the definition of random variable X_w to every wedge w in W_t. Let X_w be 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 (up to 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 only one wedge in A has $X_w = 1$.

Lemma 2.5: $X_{\{b,c\}} = X_{\{c,a\}} = 0$. 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 $\{b, c\}$ is not in *w-list*, then by definition, $X_{\{b,c\}}$ is 0. On the other hand, if the wedge is in *e-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. ∎

From the above lemma, it follows that $\mathbf{E}[X_{\{b,c\}}] = \mathbf{E}[X_{\{c,a\}}] = 0$, 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,

$$\begin{aligned}\mathbf{E}[\widehat{T}_t] &= (\alpha^2\beta)^{-1}\mathbf{E}[\sum_{w \in W_t} X_w] \\ &= (\alpha^2\beta)^{-1} \sum_{A \in T_t} \sum_{w \in A} \mathbf{E}[X_w] \\ &= (\alpha^2\beta)^{-1} \cdot \alpha^2\beta |T_t| = |T_t|\end{aligned}$$

∎

Not only are $\widehat{T}_t$ and $\widehat{W}_t$ correct on expectation, we can also prove (weak) concentration results. This is done by bounding their variance. We need some assumptions: α, β should be large enough to ensure that enough wedges of W_t are actually in *w-list*, and there are at least as many wedges in G_t as edges. Once we prove these concentration bounds, similar bounds can be shown for $\widehat{\tau}_t$. Note that this is much trickier, since $\widehat{\tau}_t$ is the ratio of two random variables. All the proofs are quite technical, and closely follow analogous arguments in previous streaming triangles work [13]. We move all these proofs into a separate subsection, which can be ignored during a first reading.

The exact statement requires an additional parameter γ that controls the quality of the estimate.

Theorem 2.6: 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 $|W_t| \geq |E_t|$ (there are at least as many wedges in G_t as edges).

Then,

$$\Pr[|\widehat{W}_t - |W_t|| > \gamma |W_t|] < \gamma$$
$$\Pr[|\widehat{T}_t - |T_t|| > \gamma |W_t|] < \gamma$$

Using these, we can prove bounds on $\widehat{\tau}_t$.

Theorem 2.7: Assume the conditions of Thm. 2.6. $|\mathbf{E}[\widehat{\tau}_t] - \tau_t| \leq 10\gamma$ and $\Pr[|\widehat{\tau}_t - \tau_t| > 8\gamma] < 4\gamma$.

C. Proofs of concentration

The most important step is to prove a variance bound for $\widehat{W}_t$ and $\widehat{T}_t$. After this, the proofs follow from a routine application of Chebyschev's inequality.

Lemma 2.8: $\max(Var[\widehat{W}_t], Var[\widehat{T}_t]) \leq (\alpha^2\beta)^{-1}|W(G_t)| + 2\alpha^{-1}|W(G_t)|^{3/2}$.

Proof: We deal with $\widehat{W}_t$ first.

$$\begin{aligned}Var[\widehat{W}_t] &= \mathbf{E}[(\widehat{W}_t)^2] - (\mathbf{E}[\widehat{W}_t])^2 \\ &= (\alpha^2\beta)^{-2}\mathbf{E}[\sum_{w \in W_t} \sum_{x \in W_t} Y_w Y_x] - |W_t|^2\end{aligned}$$

The double summation can be split based on three cases: (i) $w = x$, (ii) w and x are disjoint (they do not share an edge), and (iii) w and x have a common edge. For convenience, we will use $\sum_w$ as shorthard for $\sum_{w \in W_t}$. We use the definition of indicator Y_w from Thm. 2.3.

$$\begin{aligned}&\mathbf{E}[\sum_w \sum_x Y_w Y_x] \\ = &\sum_w \mathbf{E}[Y_w^2] + \sum_{w \cap x = \emptyset} \mathbf{E}[Y_w Y_x] + \sum_{w \cap x \neq \emptyset} \mathbf{E}[Y_w Y_x]\end{aligned}$$

The first and second are relatively easy to deal with. Since Y_w is an indicator, $Y_w^2 = Y_w$ and $\sum_w \mathbf{E}[Y_w] = (\alpha^2\beta)|W_t|$. When $w \cap x = \emptyset$, note that Y_w and Y_x are independent. This is because we assume that $hash$ is a random function. Hence,

$$\begin{aligned}\sum_{w \cap x = \emptyset} \mathbf{E}[Y_w Y_x] &= \sum_{w \cap x = \emptyset} \mathbf{E}[Y_w]\mathbf{E}[Y_x] \leq \sum_{w,x} \mathbf{E}[Y_w]\mathbf{E}[Y_x] \\ &= (\sum_w \mathbf{E}[Y_w])^2 = (\alpha^2\beta)^2 |W_t|^2\end{aligned}$$

Now for the interesting part. Suppose $w \cap x \neq \emptyset$, so $w = \{e_1, e_2\}$ and $x = \{e_1, e_3\}$. The product $Y_w Y_x$ is 1 iff e_1, e_2, e_3 are all in *e-list* and both w and x get selected in *w-list*. The probability of this is $\alpha^3\beta^2$. How many pairs of wedges $w \cap x \neq \emptyset$ are there? This is exactly $\sum_i \binom{d_i}{3}$, where d_i is the degree of vertex i in G_t. In the following, we use the fact that the ℓ_3-norm is smaller than the ℓ_2-norm. (We also use the bound $\sum_i d^2 \leq 2|W_t|$, which follows because $|W_t| \geq |E_t| = \sum_i d_i$.)

$$\begin{aligned}\sum_{w \cap x \neq \emptyset} \mathbf{E}[Y_w Y_x] &= (\alpha^3\beta^2) \sum_i \binom{d_i}{3} \\ &\leq (\alpha^3\beta^2) \sum_i d_i^3 \\ &\leq (\alpha^3\beta^2)(\sum_i d_i^2)^{3/2} \\ &\leq 2(\alpha^3\beta^2)|W_t|^{3/2}\end{aligned}$$

Putting it all together,

$$\begin{aligned} Var[\widehat{W}_t] &\leq (\alpha^2\beta)^{-2}[(\alpha^2\beta)|W_t| + (\alpha^2\beta)^2|W_t|^2 \\ &\quad +2(\alpha^3\beta^2)|W_t|^{3/2}] - |W_t|^2 \\ &= (\alpha^2\beta)^{-1}|W_t| + 2\alpha^{-1}|W_t|^{3/2} \end{aligned}$$

Note that $\widehat{T}_t = sum_w X_w$. We apply an argument identical to that above for $Var[\widehat{T}_t]$. ∎

Thm. 2.6 follows fairly directly from the variance bound.

Proof: (of Thm. 2.6) To prove a concentration bound, we will use Chebyschev's inequality. Let $Var[\widehat{W}_t]$ be the variance of $\widehat{W}_t$. Then $\Pr[|\widehat{W}_t - \mathbf{E}[\widehat{W}_t]| > h] \leq Var[X]/h^2$. Using Lem. 2.8,

$$\begin{aligned} &\Pr[|\widehat{W}_t - |W_t|| > \gamma|W_t|] \\ \leq\ &[(\alpha^2\beta)^{-1}|W_t| + 2\alpha^{-1}|W_t|^{3/2}]/(\gamma^2|W_t|^2) \\ =\ &1/(\alpha^2\beta|W_t|\cdot\gamma^2) + 1/(\alpha|W_t|^{1/2}\cdot\gamma^2) \end{aligned}$$

Since $\alpha^2\beta|W_t| \geq 1/\gamma^6$, $\alpha|W_t|^{1/2} \geq 1/\gamma^3$. Plugging this bound in, the final probability is at most γ.

An identical argument holds for $\widehat{T}_t$. ∎

We apply a Bayes' rule argument to prove bounds of $\widehat{\tau}_t$.

Proof: (of Thm. 2.7) We have $\widehat{\tau}_t = 3\widehat{T}_t/\widehat{W}_t$ if $\widehat{W}_t \neq 0$ and 0 otherwise. Let $\mathcal{E}$ denote the event that $|\widehat{W}_t - |W_t|| \leq \gamma|W_t|$ and $|\widehat{T}_t - |T_t|| \leq \gamma|W_t|$.

Conditioned on $\mathcal{E}$,

$$3\widehat{T}_t/\widehat{W}_t \leq (3|T_t| + 3\gamma|W_t|)/(1-\gamma)|W_t| \leq (1+2\gamma)\tau_t + 6\gamma$$

Similarly, conditioned on $\mathcal{E}$ $3\widehat{T}_t/\widehat{W}_t \geq (1-2\gamma)\tau_t - 6\gamma$. By Thm. 2.6, $\Pr[\mathcal{E}] \geq 1 - 2\gamma$. This proves that $\Pr[|\widehat{\tau}_t] - \tau_t| > 8\gamma] \leq \Pr[\overline{\mathcal{E}}] \leq 2\gamma$. To bound the expectation, we simply use Bayes' rule.

$$\mathbf{E}[\widehat{\tau}_t] = \mathbf{E}[\widehat{\tau}_t|\mathcal{E}]\Pr[\mathcal{E}] + \mathbf{E}[\widehat{\tau}_t|\overline{\mathcal{E}}]\Pr[\overline{\mathcal{E}}]$$

Since $\widehat{\tau}_t \in (0,1)$, the latter term is in the range $(0, 2\gamma)$. This completes the proof. ∎

D. Implementation aspects

Our implementation is basically identical to the pseudocode presented in `estimate` and `update`. We describe a few changes for handling the choice of parameters.

As described, `estimate` takes as input a single time window length Δt. But observe that the primary data structures *e-list*, *w-list*, and X_ws are independent of this window. As a result, `estimate` can handle multiple time windows with the *same* data structure. This is critical for getting a small space algorithm that provides multi-resolution views of the edge stream.

We note that if the time window $[t - \Delta t, t]$ is too small, it is unlikely that *e-list* will have any edges from this window. For any reasonable estimate, *e-list* must have sufficiently many edges (as the assumption in Thm. 2.6 states). On the other hand, small time windows can be stored explicitly to get exact answers. The volume of data is not a concern. For this reason, our algorithm and experiments focus on time windows of sufficiently large size.

The parameters α, β are described as fixed parameters. These can be tuned automatically, if we provide a fixed space bound for *e-list* and *w-list*. We initialize $\alpha = \beta = 1$. As soon as *e-list* reaches the space bound, we halve α. We delete from *e-list* any edges whose hash value is above (the new) α, and remove all wedges from *w-list* that include these edges. When *w-list* reaches its space bound, we halve β, and perform a similar cleaning. This is completely equivalent to the original algorithm.

We do not need to maintain any time histories for the edges in *e-list*, but we maintain the latest timestamp for each edge. This is necessary to check if some $w \in$ *w-list* actually formed in the time window $[t - \Delta t, t]$.

III. Empirical Results.

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 apply `estimate` on a variety of real-world datasets, both for the sake of data analysis and validation. Refer to Tab. I for details about these datasets.

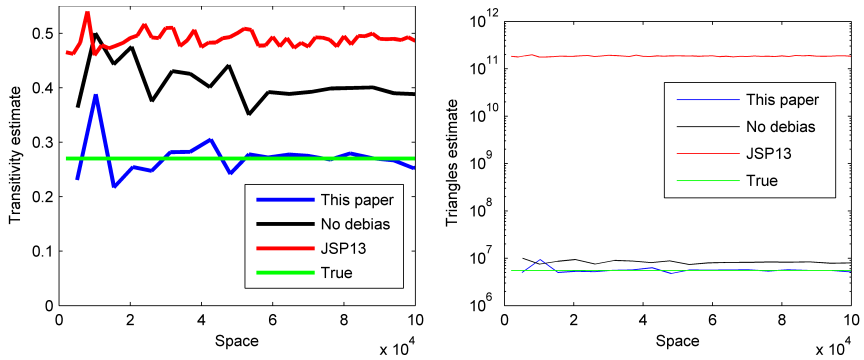
- `DBLP`: This is our primary workhorse. It is a co-authorship network generated from the metadata entry of papers on 'The DBLP Computer Science Bibliography' website. Since our approach allows working directly with the raw data in time order, we downloaded the raw XML data from DBLP [38]. We extracted 786,719 papers from the XML file where we ignored papers (i) authored by a single author, (ii) papers with more than 100 authors, and (iii) papers with 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 is the classic email network derived from emails sent between employees of Enron between 1999 and 2003. (We obtained the data from [18].) The nodes in the network correspond to the employees at Enron while edges represent their email correspondence. Naturally, multiple emails between the same pair of individuals results in the network being a multigraph.

- `Flickr`: This dataset consists of friendship connections of users of Flickr (a popular photo sharing website) which we obtained from [18]. Originally, the data was collected in the work of [19].

- Validation list: For a thorough empirical study, we have extended our data set to include networks obtained from the SNAP database [24]. We synthetically replicate edges of these datasets to get a multigraph for validation purposes.

The number of edges stored by the algorithm is measured by the total size of *e-list* and *w-list*. These are determined up to expectation by α and β, as given in Thm. 2.2. In general, we increase the total space by fixing β and increasing α (which increases the sizes of both lists). We will work with numerous time windows, either defined in terms of real-world time, or in terms of number of edges. We do not deal with very small windows, since `estimate` does not get enough samples in such a window. In any case, such windows can be handled by a brute force algorithm that stores the entire window.



(a) Transitivity (b) Triangles

Fig. 2: Comparing our algorithm with JSP13 and no debiasing for DBLP

The multigraph issue: Even ignoring time windows, we show that previous work fails on multigraphs. We run estimate and the algorithm of [13] (arguably a state-of-the-art streaming triangles algorithm, referred to as JSP13) on the DBLP graph. We gradually increase the storage up to 100K for both algorithms, and plot both the transitivity and triangles estimate. For comparison's sake, the triangles are given log-scale. *Observe how JSP13 is converging to an incorrect value of transitivity in* Fig. 2. Furthermore, the triangle estimates is off by 4 orders of magnitude. Increasing the space does not help.

We also show that the mysterious debiasing step is essential for correctness. In Fig. 2, we show the output of estimate without the debiasing (we remove Step 8 from estimate). Again, the transitivity converges to an incorrect value. The triangles estimate is also incorrect and increasing the space does not help.

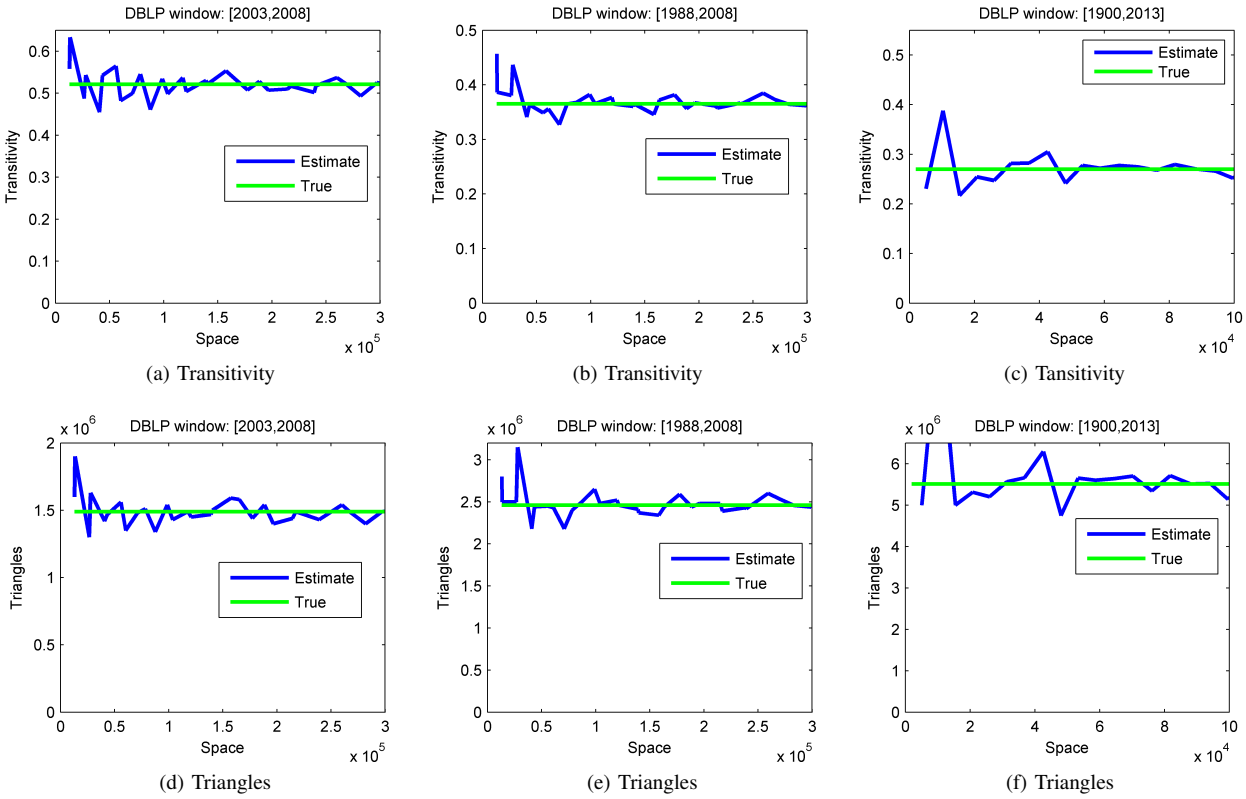
Convergence of estimate: Our estimates converge rapidly, as we demonstrate in the next series of results. We explain how a single plot is generated (Fig. 3). For DBLP, we picked specific time windows, namely, [2003,2008], [1988, 2008], and [1900, 2013] (the entire stream). This is mostly for demonstrating the convergence of differing window sizes. We fix β to 0.1 and gradually vary α in small steps to increase the space. We take the transitivity and triangles estimate for each run. We plot these results with the x-axis of increasing space (equivalently, increasing α) and the y-axis having the estimate.

Across the board, we see rapid convergence of the output at storage increases. For DBLP, storage of 200K is enough to guarantee extremely accurate results (relative errors within 5%), for all the time windows. This is even true for the 5 year window, which is quite small compared to the entire stream of data. (estimate will not work window sizes of a year, since there simply are not enough samples from such a window. But the number of edges in a year is small enough to store explicitly.)

Runs of numerous graphs: For more validation of estimate, 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 estimate to record to transitivity and triangle count of the entire stream (the graph $G[1, m]$). The results are presented in Tab. I. 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 8.6%. 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 described above), 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.

Triangle trends in DBLP: In our opinion, the following results are the real achievement of estimate. 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. 1. ("All" refers to the window that contains the entire history.) *The algorithm* estimate *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 [2003,2008] is higher than [2008,2013], 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



(a) Transitivity (b) Transitivity (c) Tansitivity

(d) Triangles (e) Triangles (f) Triangles

Fig. 3: DBLP convergence: We show that both transitivity and triangles estimates converge to true values as we increase the space available to the algorithm. The top three plots show transitivity convergence while the bottom three show triangles convergence. The plots are arranged in the order of increasing window length: 5 years, 20 years, and the entire past.

expect it to simply go to zero? Can we give a reasonable model of this behavior? We believe that the output of `estimate` will lead to many data science questions, and this is the real significance of the algorithm.

Triangle trends in `Enron`: In Fig. 4, we present triangles and transitivity estimates for `Enron` for various windows. For this dataset, we think of a window as being defined by a certain 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 will 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 sized window dominates the one corresponding to the smaller size. We observe a completely opposite behavior with `DBLP` transitivity curves, see Fig. 1.

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.

Triangle trends in `Flickr`: This is much larger dataset with 33M multiedges. We focus on time windows formed by the past 4M, 8M, 16M, and all history. These results are given in Fig. 5. We are able to get these results with merely 640K edge storage, less than 2% of the edge stream.

IV. Conclusions

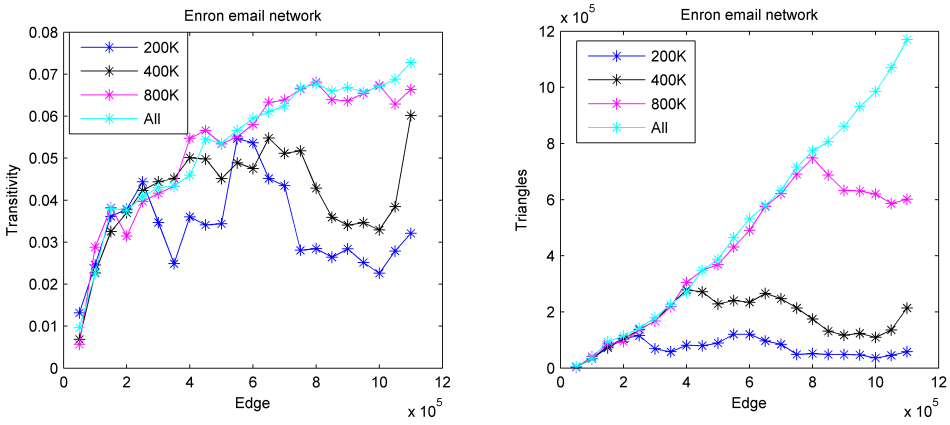
We have described a streaming algorithm to compute the number of triangles and the transitivity of a multigraph. Our algorithm can seamlessly compute estimates for stream windows of various sizes in real-time. It would be very interesting to carefully study the behavior of `estimate` on more real-world edge streams, to understand the evolution of triangles over time. It appears that `estimate` reveals phenomena at different timescales, which might be an aid to finding the "right" window for aggregation. Designing good models for temporal graphs is a big open problem, and our findings on `DBLP` and `Enron` might provide some useful information towards that objective.

Acknowledgements

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

TABLE I: A run of our algorithm on a variety of real-world and synthetic graphs. We ran the algorithm with $\alpha = 0.01$ and choosing β so that size of *w-list* is at most 50K. Here 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 [24].

Dataset	n	m	multi m	Space	Transitivity exact	Transitivity estimate	Triangles exact	Relative error in triangles estimate
DBLP	755K	254K	3.63M	31K	0.269	0.282	5.50M	3.09%
Enron	86K	297K	1.15M	8K	0.069	0.071	1.18M	3.38%
Flickr	2302K	22M	33.1M	251K	0.110	0.108	837M	1.24%
as-skitter	1.6M	11M	53M	160K	0.005	0.005	28M	6.50%
cit-Patents	3.7M	16M	79M	199K	0.067	0.066	7.51M	0.33%
web-Google	0.8M	4M	20M	79K	0.055	0.057	13.3M	3.79%
web-NotreDame	0.3M	1M	5M	42K	0.088	0.088	8.91M	3.93%
youtube	1.1M	2M	14M	64K	0.006	0.006	3.05M	1.86%
livejournal	5.2M	48M	205M	473K	0.124	0.118	310M	8.65%
orkut	3.0M	223M	562M	1.2M	0.041	0.041	627M	0.09%



(a) Transitivity (b) Triangles

Fig. 4: Given the temporal Enron stream, we use number of edges to define time windows. We run our algorithm with $\alpha = 0.04$ and $\beta = 0.4$.

the development of the final algorithm. This work was funded by the DARPA GRAPHS program. 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.

REFERENCES

[1] J. S. Coleman, "Social capital in the creation of human capital," *American Journal of Sociology*, vol. 94, pp. S95–S120, 1988. [Online]. Available: http://www.jstor.org/stable/2780243

[2] A. Portes, "Social capital: Its origins and applications in modern sociology," *Annual Review of Sociology*, vol. 24, no. 1, pp. 1–24, 1998.

[3] R. S. Burt, "Structural holes and good ideas," *American Journal of Sociology*, vol. 110, no. 2, pp. 349–399, 2004. [Online]. Available: http://www.jstor.org/stable/10.1086/421787

[4] 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*, 2010, pp. 4027–4032.

[5] J.-P. Eckmann and E. Moses, "Curvature of co-links uncovers hidden thematic layers in the World Wide Web," *Proceedings of the National Academy of Sciences (PNAS)*, vol. 99, no. 9, pp. 5825–5829, 2002.

[6] J. Berry, L. Fosvedt, D. Nordman, C. A. Phillips, and A. G. Wilson, "Listing triangles in expected linear time on power law graphs with exponent at least $\frac{7}{3}$," Sandia National Laboratories, Tech. Rep. SAND2010-4474c, 2011. [Online]. Available: http://ngc.sandia.gov/assets/documents/Berry-soda11_SAND2010-4474C.pdf

[7] D. F. Gleich and C. Seshadhri, "Vertex neighborhoods, low conductance cuts, and good seeds for local community methods," in *Knowledge Discovery and Data Mining (KDD)*, 2012.

[8] L. Becchetti, P. Boldi, C. Castillo, and A. Gionis, "Efficient semi-streaming algorithms for local triangle counting in massive graphs," in *Knowledge Data and Discovery (KDD)*, 2008, pp. 16–24.

[9] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon, "Network motifs: Simple building blocks of complex networks," *Science*, vol. 298, no. 5594, pp. 824–827, 2002.

[10] C. Seshadhri, T. G. Kolda, and A. Pinar, "Community structure and scale-free collections of Erdös-Rényi graphs," *Physical Review E*, vol. 85, no. 5, p. 056109, May 2012.

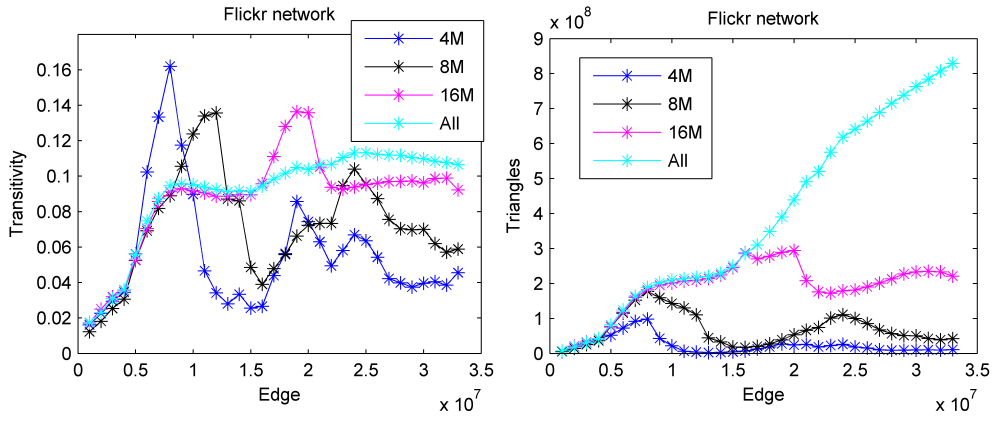
[11] N. Durak, A. Pinar, T. G. Kolda, and C. Seshadhri, "Degree relations of triangles in real-world networks and graph models," in *CIKM'12*, 2012.

[12] L. S. Buriol, G. Frahling, S. Leonardi, A. Marchetti-Spaccamela, and C. Sohler, "Counting triangles in data streams," in *Principles of Database Systems*, 2006, pp. 253–262.

[13] M. Jha, C. Seshadhri, and A. Pinar, "A space efficient streaming algorithm for triangle counting using the birthday paradox," in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, ser. KDD '13. New York, NY, USA: ACM, 2013, pp. 589–597. [Online]. Available: http://doi.acm.org/10.1145/2487575.2487678

[14] N. Ahmed, J. Neville, and R. Kompelle, "Network sampling: From static to streaming graphs," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 2013, to appear.

[15] A. Pavan, K. Tangwongsan, S. Tirthapura, and K.-L. Wu, "Counting and sampling triangles from a graph stream," in *International Conference on Very Large Databases (VLDB)*, 2013.



(a) Transitivity (b) Triangles

Fig. 5: Similar to the Enron experiments, we use number of edges to define time windows. The algorithm is run with $\alpha = \beta = 0.02$.

[16] K. Tangwongsan, A. Pavan, and S. Tirthapura, "Parallel triangle counting in massive streaming graphs," in *ACM Conference on Information & Knowledge Management (CIKM)*, 2013.

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

[18] R. Rossi and N. Ahmed, "Network repository," 2013. [Online]. Available: http://networkrepository.com

[19] A. Mislove, H. S. Koppula, K. P. Gummadi, P. Druschel, and B. Bhattacharjee, "Growth of the flickr social network," in *Proceedings of the 1st ACM SIGCOMM Workshop on Social Networks (WOSN'08)*, August 2008.

[20] S. Macskassy, "Mining dynamic networks: The importance of preprocessing on downstream analytics," in *Proceedings of 2nd International Workshop on Mining Communities and People Recommenders (COMMPER)*, 2012.

[21] G. Cormode and S. Muthukrishnan, "Space efficient mining of multigraph streams," in *PODS*, C. Li, Ed. ACM, 2005, pp. 271–282.

[22] T. Schank and D. Wagner, "Approximating clustering coefficient and transitivity," *Journal of Graph Algorithms and Applications*, vol. 9, pp. 265–275, 2005.

[23] C. Seshadhri, A. Pinar, and T. G. Kolda, "Fast triangle counting through wedge sampling," in *Proceedings of the SIAM Conference on Data Mining*, 2013. [Online]. Available: http://arxiv.org/abs/1202.5230

[24] SNAP, "Stanford network analysis project," 2013, available at http://snap.stanford.edu/.

[25] S. Venkataraman, D. X. Song, P. B. Gibbons, and A. Blum, "New streaming algorithms for fast detection of superspreaders," in *NDSS*. The Internet Society, 2005.

[26] C. E. Tsourakakis, U. Kang, G. L. Miller, and C. Faloutsos, "Doulion: counting triangles in massive graphs with a coin," in *Knowledge Data and Discovery (KDD)*, 2009, pp. 837–846.

[27] S. Suri and S. Vassilvitskii, "Counting triangles and the curse of the last reducer," in *World Wide Web (WWW)*, 2011, pp. 607–614.

[28] T. Plantenga, "Inexact subgraph isomorphism in mapreduce," *Journal of Parallel and Distributed Computing*, no. 0, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0743731512002559

[29] S. M. Arifuzzaman, M. Khan, and M. Marathe, "Patric: A parallel algorithm for counting triangles and computing clustering coefficients in massive networks," NDSSL, Tech. Rep. 12-042, 2012.

[30] D. R. Chakrabarti, P. Banerjee, H.-J. Boehm, P. G. Joisha, and R. S. Schreiber, "The runtime abort graph and its application to software transactional memory optimization," in *International Symposium on Code Generation and Optimization*, 2011, pp. 42–53. [Online]. Available: http://dl.acm.org/citation.cfm?id=2190025.2190052

[31] Y.-J. Chiang, M. T. Goodrich, E. F. Grove, R. Tamassia, D. E. Vengroff, and J. S. Vitter, "External-memory graph algorithms," in *Symposium on Discrete Algorithms (SODA)*, 1995, pp. 139–149. [Online]. Available: http://dl.acm.org/citation.cfm?id=313651.313681

[32] L. Arge, M. Goodrich, and N. Sitchinava, "Parallel external memory graph algorithms," in *Parallel Distributed Processing Symposium (IPDPS)*, april 2010, pp. 1 –11.

[33] J. Berry, B. Hendrickson, S. Kahan, and P. Konecny, "Software and algorithms for graph queries on multithreaded architectures," in *Parallel and Distributed Processing Symposium (IPDPS)*, march 2007, pp. 1 – 14.

[34] H. Jowhari and M. Ghodsi, "New streaming algorithms for counting triangles in graphs," in *Computing and Combinatorics Conference (COCOON)*, 2005, pp. 710–716.

[35] K. J. Ahn, S. Guha, and A. McGregor, "Graph sketches: sparsification, spanners, and subgraphs," in *Principles of Database Systems*, 2012, pp. 5–14.

[36] D. M. Kane, K. Mehlhorn, T. Sauerwald, and H. Sun, "Counting arbitrary subgraphs in data streams," in *International Colloquium on Automata, Languages, and Programming (ICALP)*, 2012, pp. 598–609.

[37] M. A. Hasan, N. Ahmed, and J. Neville, "Network sampling: Methods and applications," in *Proceedings of ACM SIGKDD*, 2013, http://www.cs.purdue.edu/homes/neville/courses/kdd13-tutorial.html.

[38] M. Ley, "Digital bibliography & library project," available at http://www.informatik.uni-trier.de/~ley/db/.