A Space Efficient Streaming Algorithm for Triangle Counting using the Birthday Paradox*

[Extended Abstract]

Madhav Jha†
Pennsylvania State University
State College, PA, USA
mxj201@psu.edu

C. Seshadhri
Sandia National Labs
Livermore, CA, USA
scomand@sandia.gov

Ali Pinar
Sandia National Labs
Livermore, CA, USA
apinar@sandia.gov

ABSTRACT

We design a space efficient algorithm that approximates the transitivity (global clustering coefficient) and total triangle count with only a single pass through a graph given as a stream of edges. Our procedure is based on the classic probabilistic result, *the birthday paradox*. When the transitivity is constant and there are more edges than wedges (common properties for social networks), we can prove that our algorithm requires $O(\sqrt{n})$ space (n is the number of vertices) to provide accurate estimates. We run a detailed set of experiments on a variety of real graphs and demonstrate that the memory requirement of the algorithm is a tiny fraction of the graph. For example, even for a graph with 200 million edges, our algorithm stores just 60,000 edges to give accurate results. Being a single pass streaming algorithm, our procedure also maintains a real-time estimate of the transitivity/number of triangles of a graph, by storing a miniscule fraction of edges.

Categories and Subject Descriptors

F.2.2 [**Nonnumerical Algorithms and Problems**]: Computations on Discrete Structures; G.2.2 [**Graph Theory**]: Graph Algorithms

General Terms

Algorithms, Theory

*This work was funded by the DOE ASCR Complex Interconnected Distributed Systems (CIDS) program and Sandia's Laboratory Directed Research & Development (LDRD) 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.

†Work done while the author was interning at Sandia National Labs, Livermore.

KDD'13, August 11–14, 2013, Chicago, Illinois, USA.

Keywords

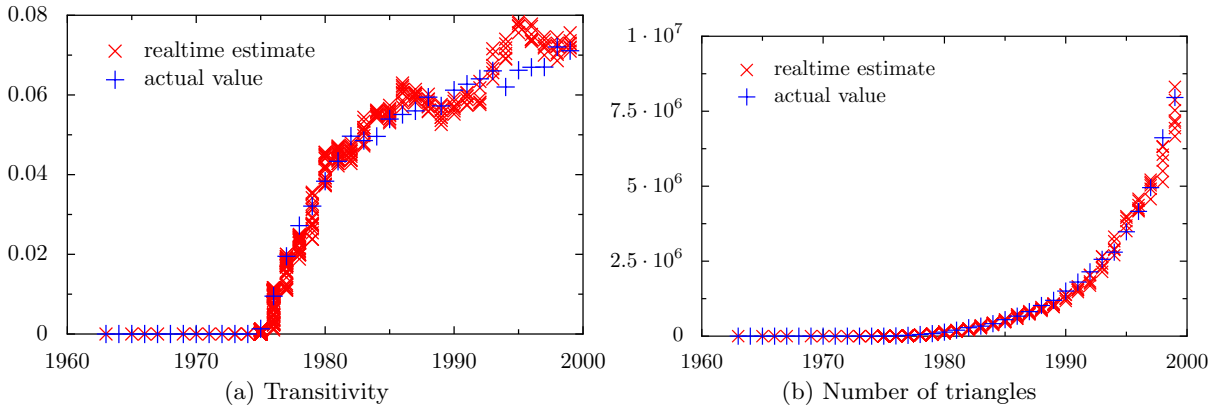
Streaming algorithms, Triangle counting

1. INTRODUCTION

Triangles are one of the most important motifs in real world networks. Whether the networks come from social interaction, computer communications, financial transactions, proteins, or ecology, the abundance of triangles is pervasive and it is critical feature that distinguishes real graphs from random graphs. There is a rich body of literature on analysis of triangles and counting algorithms. Social scientists use triangle counts to understand graphs [17, 28, 11, 39]; graph mining applications such as spam detection and finding common topics on the WWW use triangle counts [18, 7]; motif detection in bioinformatics often count the frequency of triadic patterns [25]. Nevertheless, counting triangles continues to be a challenge due to sheer size of the graphs (easily in the order of billions of edges).

Many massive graphs come from modeling interactions in a dynamic system. People call each other on the phone, exchange emails, or become part of a tight unit (e.g, coauthoring a paper); computers exchange messages; animals come in the vicinity of each other; companies trade with each other. These interactions manifest as a *stream of edges*. The edges appear with timestamps, or "one at a time." The network (graph) that represents the system is an accumulation of the observed edges. There are many methods to deal with such massive graphs, such as random sampling [29, 36, 31], MapReduce paradigm [32, 27], distributed-memory parallelism [4, 12], adopting external memory [13, 3], and multithreaded parallelism [9].

But all of these methods need to store at least a large fraction of the data. A *small space streaming algorithm* maintains a very small (using some randomness) set of edges, called the "sketch" at any given time, and updates this sample as edges appear. Based on 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 sketch size is orders of magnitude smaller than the total graph. Furthermore, it can be updated rapidly when new edges arrive, and hence maintains a real-time estimate of the number of triangles. We also want a single pass algorithm, so it only observes each edge once (think of it as making a single scan of a log file). The algorithm cannot revisit edges that it has forgotten.



(a) Transitivity (b) Number of triangles

Figure 1: Realtime tracking of number of triangles and transitivities on cit-Patents (16M edges), storing only 60K edges from the past. (Absolute values are shown with time (in years) on the x-axis.)

1.1 The streaming setting

Let G be a simple undirected graph with n vertices and m edges. Let T denote the number of triangles in the graph and W be the number of *wedges*, where a *wedge* is a path of length 2. A common measure is the *transitivity* $\kappa = 3T/W$ [38], a measure of how often friends of friends are also friends. (This is also called the global clustering coefficient [31].)

Formally, a single pass streaming algorithm is defined as follows. Consider a sequence of (distinct) edges $e_1, e_2, \ldots, e_m$. Let G_t be the *graph at time* t, formed by the edge set $\{e_i | i \leq t\}$. The stream of edges can be thought of as a sequence of *edge insertions* into the graph. (Vertex insertions can be trivially handled.) We do not know the number of vertices ahead of time, and simply see each edge as a pair (u, v) of vertex labels. So new vertices are implicitly added as new labels. There is no assumption on the order of edges in the stream. Edges incident to a single vertex do not necessarily appear together.

In this paper, we do not consider edge/vertex deletions or repeated edges. In that sense, this is a simplified version of the full-blown streaming model. Nonetheless, previous work on counting triangles focuses primarily of this model [6, 21, 10, 1, 22].

A streaming algorithm has a small memory M, and sees the edges in stream order. At each edge e_t, the algorithm can choose to update data structures in M (using the edge e_t). Then the algorithm proceeds to e_{t+1}, and so on. The algorithm is never allowed to recall an edge that has already passed by. The memory M is much smaller than m, so the algorithm keeps a small "sketch" of the edges it has seen. The aim is to estimate the number of triangles in $G := G_m$ at the end of the stream. Usually, we desire the more stringent guarantee of maintaining a running estimate of triangles and transitivity of G_t at time t. We denote these quantities respectively as T_t and κ_t.

1.2 Results

We present a single pass, $O(m/\sqrt{T})$-space algorithm to provably estimate the transitivity (with arbitrary additive error) in a streaming graph. Streaming algorithms for counting triangles or computing the transitivity have been studied before, but no previous algorithm attains this space guarantee. (Buriol et al [10] give a single pass algorithm with a stronger relative error guarantee that requires space $O(mn/T)$. We discuss in more detail later.)

Although our theoretical result is interesting asymptotically, the constant factors and dependence on error in our bound are large. Our main result is a *practical* streaming algorithm (based on the theoretical one) for computing κ and T, using some additional probabilistic heuristics. We perform an extensive empirical analysis of our algorithm on a variety of datasets (many thanks to SNAP [41], for their extensive collection of downloadable graphs). The salient features of our algorithm are:

- **Theoretical basis:** Our algorithm is based on the classic *birthday paradox*: if we choose 23 random people, the probability that 2 of them share a birthday is at least 1/2 (Chap. II.3 of [19]). We extend this analysis for sampling wedges in a large pool of edges. The final streaming algorithm is designed by using reservoir sampling with wedge sampling [31] for estimating κ. We prove a space bound of $O(m/\sqrt{T})$, which we show is $O(\sqrt{n})$ under common conditions for social networks.

While our theory appears to be a good guide in designing the algorithm and explaining its behavior, it should not be used to actually decide space bounds in practice. Also, we point out that for graphs where T is small, our algorithm does not provide good guarantees for small space (since $m/\sqrt{T}$ is large).

- **Accuracy and scalability with small sketches:** We test our algorithm on a variety of graphs from different sources. In all instances, we get accurate estimates for κ and T by storing at most 40K edges. This is even for graphs where m is in the order of millions. Our relative errors on κ and the number of triangles are mostly less than 5%. (In a graph with very few triangles where $\kappa < 0.01$, our triangle estimate has relative error of 12%.) Our algorithm processes extremely large graphs. Our experiments include a run on a streamed Orkut social network with 200M edges (by storing only 40K edges, relative errors are at most 4%). We get similar results on streamed Flickr and Live-journal graphs with tens of millions of edges.

We run detailed experiments on some test graphs (with 1-3 million edges) with varying parameters to show convergence of our algorithm. Comparisons with previous work [10] show

that our algorithm gets within 5% of the true answer, while the previous algorithm is off by more than 50%.

• **Real-time tracking:** For a temporal graph, our algorithm precisely tracks both κ_t and T_t with less storage. By storing 60K edges of the past, we can track this information for a patent citation network with 16 million edges [41]. Refer to Fig. 1. We maintain a real-time estimate of both the transitivity and number of triangles with a single pass, storing less than 1% of the graph. We see some fluctuations in the transitivity estimate due to the randomness of the algorithm, but the overall tracking is fairly accurate.

2. PREVIOUS WORK

Enumeration of all triangles is a well-studied problem [14, 30, 24, 16, 8, 15, 32, 4]. Eigenvalue/trace based methods have also been used [33, 5] to compute estimates of the total and per-degree number of triangles. A long line of work on sparsification methods [23, 35, 40, 26] started with Tsourakakis et al. [34].

Theoretical streaming algorithms for counting triangles were initiated by Bar-Yossef et al. [6]. Subsequent improvements were given in [21, 10, 1, 22]. The space bounds achieved are of the form mn/T. Note that $m/\sqrt{T} \leq mn/T$ whenever $T \leq n^2$ (which is a reasonable assumption for sparse graphs). These algorithm are rarely practical, since T is often much smaller than mn. (Some multi-pass streaming algorithms give stronger guarantees, but we will not discuss them here.)

Buriol et al. [10] give an implementation of their algorithm. For almost all of their experiments on graphs, with storage of 100K edges, they get fairly large errors (always more than 10%, and often more than 50%). Buriol et al provide an implementation in the incidence list setting, where all neighbors of a vertex arrive together. In this case, their algorithm is quite pratical since the errors are quite small. Our algorithm scales to sizes (100 million edges) larger than their experiments. We get better accuracy with far less storage, without any assumption on the ordering of the data stream. Furthermore, our algorithm performs accurate real-time tracking. Becchetti et al. [7] gave a semi-streaming algorithm for counting the triangles incident to *every* vertex. Their algorithm uses clever methods to approximate Jaccard similarities, and requires multiple passes over the data.

Wedge-sampling based algorithms for various triadic measures on graphs [29, 31] work by sampling random wedges of the graph. These algorithms are extremely accurate, but the random sampling require the entire graph to be present. Our algorithm can be thought of as a streaming variant of wedge-sampling. The birthday paradox argument has been connected to triangle counts, but in an entirely different context. Alon et al [2] use this to prove *lower bounds* for finding triangles in graphs.

3. OUTLINE

We begin in §4 by providing an intuitive high-level explanation of the algorithm. §5 gives a formal description of our implemented algorithm (denoted Streaming-Triangles). The theoretical analysis is done in §6, for an idealized (and inefficient) variant called Single-Bit. We stress that Single-Bit is a thought experiment to highlight the theoretical aspects of our result, and we do not actually implement it. In §6.1, we explain the heuristics used to get Streaming-Triangles. The remainder of §6 gives a mathematical analysis of Single-Bit.

Finally, in §7, we give results on our runs of Streaming-Triangles on real graphs.

4. INTUITION FOR THE ALGORITHM

The starting point is the idea of *wedge sampling* to estimate κ [31]. A wedge is *closed* if it participates in a triangle and *open* otherwise. Note that $\kappa = 3T/W$ is exactly the probability that a uniform random wedge is closed. This gives a simple randomized algorithm for estimating κ (and T), by generating a set of (independent) uniform random wedges and finding the fraction that are closed. But how do we sample wedges from a stream of edges?

Suppose we just sampled a uniform random set of edges. How large does this set need to be to get a wedge? The *birthday paradox* can be used to deduce that (as long as $W \geq m$, pretty much always true for real networks) $O(\sqrt{n})$ edges suffice. A more sophisticated result, given in Lem. 1, provides (weak) concentration bounds on the number of wedges generated by a random set of edges. A "small" number of uniform random edges can give enough wedges to perform wedge sampling (which in turn is used to estimate κ).

A set of uniform random edges can be maintained by the standard method of *reservoir sampling* [37]. From these edges, we generate a random wedge by doing a second level of reservoir sampling. This implicitly treats the wedges created in the edge reservoir as a stream, and performs reservoir sampling on that. Overall, this approximates uniform random wedge sampling.

As we maintain our reservoir wedges, we check for closure by the future edges in the stream. But there are closed wedges that cannot be verified, because the closing edge may have already appeared in the past. A simple observation used by past streaming algorithms saves the day [21, 10]. In each triangle, there is exactly one wedge whose closing edge appears in the future. So we try to approximate the fraction of these "future closed" wedges, which is exactly one-third of the fraction of closed wedges. (Hence, the factor 3 that pops up in Streaming-Triangles.)

Finally, to estimate T from κ, we need an estimate on the total number of wedges W. This can be obtained by reverse engineering the birthday paradox: given the number of wedges in our reservoir sample of edges, we can estimate W (again, using the workhorse Lem. 1).

5. THE PROCEDURE Streaming-Triangles

The streaming algorithm maintains two primary sets of data: the *edge reservoir* and the *wedge reservoir*. These are sets of edges and wedges that the algorithm stores from the past. The parameters for the streaming algorithm are the respective sizes of these sets, denoted by s_e and s_w respectively. The main algorithm is described in Streaming-Triangles, although most of the technical computation is performed in Update (which is invoked every time a new edge appears). After processing edge e_t, the algorithm computes running estimates for κ_t and T_t. These values do not have to be stored, so they are immediately output and forgotten. We describe the main data structures of the algorithm Streaming-Triangles.

- Array *edge_res*$[1 \cdots s_e]$: This is the array of *reservoir edges* and is the primary subsample of the stream maintained.
- New wedges $\mathcal{N}_t$: This is a list of all wedges involving e_t formed only by edges in *edge_res*. This may often be empty, if e_t is not in *edge_res*. We do not necessarily maintain this list explicitly, and we discuss implementational details later.
- Variable *tot_wedges*: This is the total number of wedges formed by edges in *edge_res*.
- Array *wedge_res*$[1 \cdots s_w]$: This is an array of *reservoir wedges* of size s_w.
- Array *isClosed*$[1 \cdots s_w]$: This is a boolean array. We set *isClosed*$[i]$ to be true if wedge *wedge_res*$[i]$ is found to be closed.

Algorithm 1: STREAMING-TRIANGLES(s_e, s_w)

Initialize *edge_res* of size s_e and *wedge_res* of size s_w.
For each edge e_t in stream,
  Call UPDATE(e_t).
  Let ρ be the fraction of entries in *isClosed* set to `true`.
  Set $\kappa_t = 3\rho$.
  Set $T_t = [\rho t^2/s_e(s_e - 1)] \times tot\_wedges$.

Algorithm 2: UPDATE(e_t)

For every reservoir wedge *wedge_res*$[i]$ closed by e_t, set *isClosed*$[i]$ = `true`.
Flip a coin with heads probability $= 1 - (1 - 1/t)^{s_e}$. If it flips to tails, stop and proceed to the next edge in stream. (So remaining code is processed only if coin comes heads.)
Choose a uniform index $i \in [s_e]$. Set *edge_res*$[i] = e_t$.
Determine $\mathcal{N}_t$ and let *new_wedges* $= |\mathcal{N}_t|$.
Update *tot_wedges*, the number of wedges formed by *edge_res*.
Set $q =$ *new_wedges*/*tot_wedges*.
For each index $i \in [s_w]$,
  Flip coin with heads probability q.
  If tails, continue to next index in loop.
  Pick uniform random $w \in \mathcal{N}_t$ that involves e_t.
  Replace *wedge_res*$[i] = w$. Reset *isClosed*$[i]$ = `false`.

5.1 Implementation details

Computing κ_t and T_t are simple and require no overhead. We maintain *edge_res* as a time-variable subgraph. Each time *edge_res* is updated, the subgraph undergoes an edge insert and edge delete. Suppose $e_t = (u, v)$. Wedges in $\mathcal{N}_t$ are given by the neighbors of u and v in this subgraph. From random access to the neighbor lists of u and v, we can generate a random wedge from $\mathcal{N}_t$ efficiently.

Updates to *edge_res* are very infrequent. At time t, the probability of an update is $1 - (1 - 1/t)^{s_e}$. By linearity of expectation, the total number of times that $\mathcal{N}_t$ is non-empty is

$$\sum_{t \le m} 1 - (1 - 1/t)^{s_e} \approx \sum_{t \le m} s_e/t \approx s_e \ln m$$

For a fixed s_e, this increases very slowly with m. So for most steps, we neither update *edge_res* or sample a new wedge.

The total number of edges that are stored from the past is $s_e + s_w$. The edge reservoir explicitly stores edges, and at most s_w edges are implicitly stored (for closure). Regardless of the implementation, the extra data structures overhead is at most twice the storage parameters s_e and s_w. Since these are at least 2 orders of magnitude smaller than the graph, this overhead is easy to pay for.

6. THE IDEALIZED ALGORITHM SINGLE-BIT

The algorithm called SINGLE-BIT is an idealized variant of STREAMING-TRIANGLES that we can formally prove theorems about. It requires more memory and expensive updates, but explains the basic principles behind our algorithm. We later give the memory reducing heuristics that take us from SINGLE-BIT to STREAMING-TRIANGLES.

The procedure SINGLE-BIT has a single space parameter s and outputs a single (random) bit at each t. The expectation of this bit is related to the transitivity κ_t. We will describe SINGLE-BIT in a more mathematical fashion. SINGLE-BIT maintains a set of edges $\mathcal{R}_t$ of size s. The set of wedges constructed from $\mathcal{R}_t$ is $\mathcal{W}_t$. Formally, $\mathcal{W}_t = \{\text{wedge } (e, e') | e, e' \in \mathcal{R}_t\}$. For every e_t, SINGLE-BIT records all wedges closed by this edge. SINGLE-BIT maintains a set $\mathcal{C}_t$, the set of wedges in $\mathcal{W}_t$ for which it has *detected a closing edge*. (Note that this is a subset of all closed wedges in $\mathcal{W}_t$.) This set is easy to update as $\mathcal{R}_t$ changes.

Algorithm 3: SINGLE-BIT(s)

Initialize $\mathcal{R}_0$ as a set of s dummy edges.
For each e_t in stream,
  For each edge in $\mathcal{R}_{t-1}$, replace it (independently) by e_t with probability $1/t$. This yields $\mathcal{R}_t$.
  Construct the set of wedges $\mathcal{W}_t$.
  Let $\mathcal{C}_t$ be the set of all wedges in $\mathcal{W}_t$ closed by e_t
  Add all wedges in $\mathcal{C}_{t-1}$ to $\mathcal{C}_t$. If $\mathcal{W}_t$ is empty, output $b_t = 0$ and continue to next edge.
  Pick a uniform random wedge in $\mathcal{W}_t$.
  Output $b_t = 1$ if this wedge is in $\mathcal{C}_t$ and $b_t = 0$ otherwise.

For convenience, we state this theorem for the final time step. However, it also holds (with an identical proof) for any large enough time t.

THEOREM 1. *Suppose $s \ge cm/(\beta^3\sqrt{T})$, for some sufficiently large constant c. Set $est = m^2|\mathcal{W}_m|/(s(s-1))$. Then $|\kappa/3 - \mathbf{E}[b_m]| < \beta$ and with probability $> 1 - \beta$, $|W - est| < \beta W$.*

We now show that $m/\sqrt{T} = O(\sqrt{n/\kappa})$ (usually much smaller for heavy tailed graphs) when $W \ge m$. Denote the degree of vertex v by d_v. In this case, we can bound $2W = \sum_v d_v(d_v - 1) = \sum_v d_v^2 - 2m \ge \sum_v d_v^2 - 2W$, so $W \ge \sum_v d_v^2/4$. By $2m = \sum_v d_v$ and the Cauchy-Schwartz inequality,

$$\frac{m}{\sqrt{W}} \le \frac{\sum_v d_v}{\sqrt{\sum_v d_v^2}} \le \frac{\sqrt{\sum_v 1}\sqrt{\sum_v d_v^2}}{\sqrt{\sum_v d_v^2}} = \sqrt{n}$$

Using the above bound, we get $m/\sqrt{T} = \sqrt{3}m/\sqrt{\kappa W} \leq \sqrt{3n/\kappa}$. Hence, when $W \geq m$ and κ is a constant (both reasonable assumptions for social networks), we require only $O(\sqrt{n})$ space.

6.1 Circumventing problems with SINGLE-BIT

SINGLE-BIT has two main problems, which are handled in STREAMING-TRIANGLES by heuristic arguments.

Thm. 1 immediately gives a small sublinear space streaming algorithm for estimating κ. The output of SINGLE-BIT has the right expectation. We can run many independent invocations of SINGLE-BIT and take the fraction of 1s output to estimate $\mathbf{E}[b_m]$ (which in turn is close to $\kappa/3$). A Chernoff bound tells us that $O(1/\epsilon^2)$ invocations suffice to estimate $\mathbf{E}[b_m]$ within additive error ϵ. The total space becomes $O(m/(\sqrt{T}\epsilon^2))$, which can be very expensive in practice. Even though $m/\sqrt{T}$ is not large, for the reasonable value of $\epsilon = 0.01$, the storage cost blows up by a factor of 10^4. This is the standard method used in previous work for streaming triangle counts.

This blowup is avoided in STREAMING-TRIANGLES by *reusing the same reservoir of edges* for sampling wedges. Note that SINGLE-BIT is trying to generate a single uniform random wedge from G, and we use independent reservoirs of edges to generate multiple samples. Lem. 1 says that for a reservoir of $km/\sqrt{W}$ edges, we expect k^2 wedges. So, if $k > 1/\epsilon$ and we get $> 1/\epsilon^2$ wedges. Since the reservoir contains a large set of wedges, we could just use a subset of these for estimating $\mathbf{E}[b_m]$. Unfortunately, these wedges are correlated with each other, and we cannot theoretically prove the desired concentration. In practice, it appears that these wedges are sufficiently uncorrelated that we get excellent results by reusing the reservoir. This is an important distinction from past streaming work [21, 10]. We can multiply our space by $1/\epsilon$ to (heuristically) get error ϵ, but this is not possible through previous algorithms. Their space is multiplied by $1/\epsilon^2$.

The second issue is that SINGLE-BIT requires a fair bit of bookkeeping. We need to generate a random wedge from the large set $\mathcal{W}_t$. While this is possible by storing *edge_res* as a subgraph, we have a nice (at least in the authors' opinion) heuristic fix that avoids these complications.

Suppose we have a uniform random wedge $w \in \mathcal{W}_{t-1}$. We can convert it to an "almost" uniform random wedge in $\mathcal{W}_t$. If $\mathcal{W}_t = \mathcal{W}_{t-1}$ is true (so $\mathcal{N}_t = \emptyset$ which is true most of of the time), then w is also uniform in $\mathcal{W}_t$. Suppose not. Note that $\mathcal{W}_t$ is constructed by removing some wedges from $\mathcal{W}_{t-1}$ and inserting $\mathcal{N}_t$. Since w is uniform random in $\mathcal{W}_{t-1}$, *if w is also present in $\mathcal{W}_t$*, then it is uniform random in $\mathcal{W}_t \setminus \mathcal{N}_t$. Replacing w by a uniform random wedge in $\mathcal{N}_t$ with probability $|\mathcal{N}_t|/|\mathcal{W}_t|$ yields a uniform random wedge in $\mathcal{W}_t$. This is precisely what STREAMING-TRIANGLES does.

When $w \notin \mathcal{W}_t$, then the edge replaced by e_t must be in w. We approximte this as a low probability event and simply ignore this case. Hence, in STREAMING-TRIANGLES, we simply assume that w is always in $\mathcal{W}_t$. This is technically incorrect, it appears to have little effect on the accuracy in practice. And it leads to a cleaner, efficient implementation.

6.2 Proving Thm. 1

Due to space considerations and for clarity's sake, we provide proof sketches in this version. Mathematical details can be found in the online full version [20].

We begin with some preliminaries. First, the set $\mathcal{R}_t$ is a set of s uniform i.i.d. samples from $\{e_1, e_2, \ldots, e_t\}$, a direct consequence of reservoir sampling. Next, we define *future-closed* wedges. Take the final graph G and label all edges with their timestamp. For each triangle, the wedge formed by the earliest two timestamps is a *future-closed wedge*. In other words, if a triangle T has edges e_i, e_j, e_k, $(i < j < k)$, then the wedge $\{e_i, e_j\}$ is future-closed. The number of future-closed wedges is exactly T, since each triangle contains a single such wedge. We now have a simple yet important claim about the output of SINGLE-BIT.

CLAIM 1. *The set $\mathcal{C}_m$ is exactly the set of future closed wedges in $\mathcal{W}_m$.*

PROOF. Consider some wedge $\{e_i, e_j\}, i < j$ in $\mathcal{W}_m$. This wedge was formed at time j, and remains in all $\mathcal{W}_t$ for $j \leq t \leq m$. If this wedge is future closed (say by edge $e_{t'}$, for $t' > j$), then at time t', the wedge will be detected to be closed. Since this information is maintained by SINGLE-BIT, the wedge will be in $\mathcal{C}_m$. If the wedge is not future closed, then no closing edge will be found for it after time j. Hence, it will not be in $\mathcal{C}_m$. □

Consider the situation of the data structures at the end. The main technical effort goes into showing that the number of wedges formed (by edges) in $\mathcal{R}_m$ (this is precisely $|\mathcal{W}_m|$) can be used to determine the actual number of wedges in G_m. Furthermore, the number of future closed wedges in $\mathcal{R}_m$ (precisely $|\mathcal{C}_m|$, by Claim 1) can be used to estimate T.

This is formally expressed in the next lemma. Roughly, if $s = km/\sqrt{W}$, then we expect k^2 wedges formed by $\mathcal{R}_m$. We also get weak concentration bounds for the quantity. A similar bound (with somewhat weaker concentration) holds even when we consider the set of future closed wedges.

LEMMA 1 (BIRTHDAY PARADOX FOR WEDGES). *Let G be a graph with m edges and $\mathcal{S}$ be some fixed subset of wedges in G. Let $\mathcal{R}$ be a set of s uniformly and independently selected edges (with replacement) from G. Let X be the random variable denoting the number of wedges in $\mathcal{S}$ formed by edges in $\mathcal{R}$.*

1. $\mathbf{E}[X] = \binom{s}{2}(2|\mathcal{S}|/m^2)$.
2. *Let $\beta > 0$ be a parameter. If $s \geq 3m/(\beta^3\sqrt{|\mathcal{S}|})$, then with probability at least $1 - \beta$, $|X - \mathbf{E}[X]| \leq (\beta W/|\mathcal{S}|)\mathbf{E}[X]$.*

Proof sketch. The first part is an adaptation of the birthday paradox calculation. Let the set $\mathcal{R} = \{r_1, r_2, \ldots, r_s\}$. We define random variables $X_{i,j}$ for each $i, j \in [s]$ with $i < j$. Let $X_{i,j} = 1$ if the wedge $\{r_i, r_j\}$ belongs to S and 0 otherwise. Then $X = \sum_{i<j} X_{i,j}$.

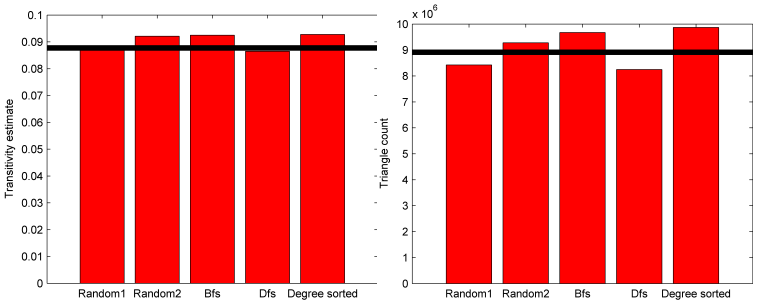
Since $\mathcal{R}$ consists of uniform i.i.d. edges from G, the following holds: for every $i < j$ and every (unordered) pair of edges $\{e_\alpha, e_\beta\}$ from E, $\Pr[\{r_i, r_j\} = \{e_\alpha, e_\beta\}] = 2/m^2$. This implies $\Pr[X_{i,j} = 1] = 2|\mathcal{S}|/m^2$. By linearity of expectation, we have $\mathbf{E}[X] = \binom{s}{2}\mathbf{E}[X_{i,j}] = \binom{s}{2}\Pr[X_{i,j} = 1] = \binom{s}{2}(2|\mathcal{S}|/m^2)$, as required.

The second part is obtained by bounding the variance of X and using the Chebyschev inequality. It is a fairly technical proof that can be found in the full version. □

We now sketch the proof of Thm. 1. (As mentioned earlier, the complete rigorous proof can be found in the full

Table 1: Properties of the graphs used in the experiments

Graph	n	m	W	T	κ
amazon0312	401K	2350K	69M	3686K	0.160
amazon0505	410K	2439K	73M	3951K	0.162
amazon0601	403K	2443K	72M	3987K	0.166
as-skitter	1696K	11095K	16022M	28770K	0.005
cit-Patents	3775K	16519K	336M	7515K	0.067
DBLP	317K	1049K	21M	224K	0.3064
roadNet-CA	1965K	2767K	6M	121K	0.060
web-BerkStan	685K	6649K	27983M	64691K	0.007
web-Google	876K	4322K	727M	13392K	0.055
web-NotreDame	326K	1090K	305M	8910K	0.088
web-Stanford	282K	1993K	3944M	11329K	0.009
wiki-Talk	2394K	4660K	12594M	9204K	0.002
youtube	1158K	2990K	1474M	3057K	0.006
flickr	1861K	15555K	14670M	548659K	0.112
livejournal	5284K	48710K	7519M	310877K	0.124
orkut	3073K	223534K	45625M	627584K	0.041



(a) Transitivity (b) Number of triangles

Figure 2: Output of a single run of Streaming-Triangles on various orderings of the data stream for web-NotreDame. The true value is given by the thick black line.

version [20].) At the end of the stream, the output bit b_m is 1 if $|\mathcal{W}_m| > 0$ and a wedge from $\mathcal{C}_m$ is sampled. Note that both $|\mathcal{W}_m|$ and $|\mathcal{C}_m|$ are random variables.

To deal with the first event, we apply Lem. 1 with $\mathcal{S}$ being the set of all wedges. So, $\mathbf{E}[|\mathcal{W}_m|] = \binom{s}{2}(2W/m^2)$. Since $s \geq cm/(\beta^3\sqrt{T}) \geq cm/(\beta^3\sqrt{W})$, $\mathbf{E}[|\mathcal{W}_m|] \geq c^2/\beta^6$ (a fairly large number). Intuitively, the probability that $|\mathcal{W}_m| = 0$ should be very small, and this can be bounded using the concentration bound of Lem. 1.

For this informal discussion, let us just assume that $|\mathcal{W}_m| > 0$ happens. Now, the probability that $b_m = 1$ (which is $\mathbf{E}[b_m]$) is exactly the fraction $|\mathcal{C}_m|/|\mathcal{W}_m|$. Suppose both these behave like their expectation. By Claim 1, $\mathcal{C}_m$ is the set of future closed wedges. The number of future closed wedges is T, so Lem. 1 tells us that $\mathbf{E}[|\mathcal{C}_m|] = \binom{s}{2}(2T/m^2)$. Hence, $\mathbf{E}[|\mathcal{C}_m|]/\mathbf{E}[|\mathcal{W}_m|] = T/W = \kappa/3$.

In general, the value of $|\mathcal{C}_m|/|\mathcal{W}_m|$ might be quite different from $\mathbf{E}[|\mathcal{C}_m|]/\mathbf{E}[|\mathcal{W}_m|]$. Note that $\mathbf{E}[|\mathcal{C}_m|] \geq c^2/\beta^6$, by choice of s. Using the concentration bounds of Lem. 1, we can argue that the deviation of $|\mathcal{C}_m|/|\mathcal{W}_m|$ from $\mathbf{E}[|\mathcal{C}_m|]/\mathbf{E}[|\mathcal{W}_m|]$ is at most β with probability $> 1 - \beta$.

7. EXPERIMENTAL RESULTS

We implemented our algorithm in C++ and ran our experiments on a MacBook Pro laptop equipped with a 2.8GHz Intel core i7 processor and 8GB memory. The vital statistics of all the graphs that we experiment upon are provided in Tab. 1. For our case studies, we focus on the web-NotreDame and amazon0505 graphs, having 1M and 3M edges respectively.

Effects of storage on estimates: We explore the effect that the parameters s_e, s_w (referred to as the edge reservoir and wedge reservoir sizes) have on the quality of the estimates for κ and T. Both of these graphs have around 1M edges. Each of these is converted into a random stream of edges.

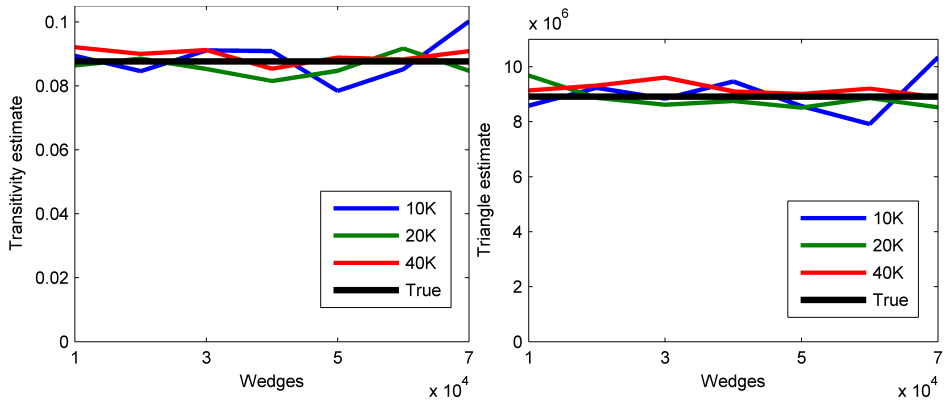
We set s_e to be 10K, 20K, and 40K, and vary s_w from 10K to 70K in increments of 10K. For each setting of the parameters, we perform a single run of Streaming-Triangles on these streamed graphs. This is to give a true indication of the behavior of Streaming-Triangles. The results are shown in Fig. 3 and Fig. 4, both for the transitivity and triangle counts for web-NotreDame and amazon0505. We fix a setting of s_e and increase s_w. In terms of the theory, a large s_e helps generate wedges that are closer to uniform. A larger s_w then helps to get a sharper estimate for the fraction of future closed wedges.

Observe that the algorithm is almost always within 10% of the true answer. The statistical deviation is larger from edge reservoirs of 10K and 20K, but it quite small for 40K. Indeed for s_e =40K, the output is well within 5%. In general, these figures show that the algorithm does not have wild statistical fluctations and is fairly well behaved as we increase the space. Increasing the space gives a predictable improvement in the estimate.

Comparison with previous work: The streaming algorithm of Buriol et al [10] was implemented and run on real graphs. In general, they get fairly large error even with storage of 100K edges. We provide comparisons with an implementation of their algorithm for arbitrary streams (they also have a version only for incidence streams, but we do not compare with that). The basic sampling procedure involves sampling a random edge and a random vertex and trying the complete a triangle. This is repeatedly independently in parallel to get an estimate for the number of triangles. Buriol et al provide various heuristics to speed up their algorithm, but the core sampling procedure is what was described above.

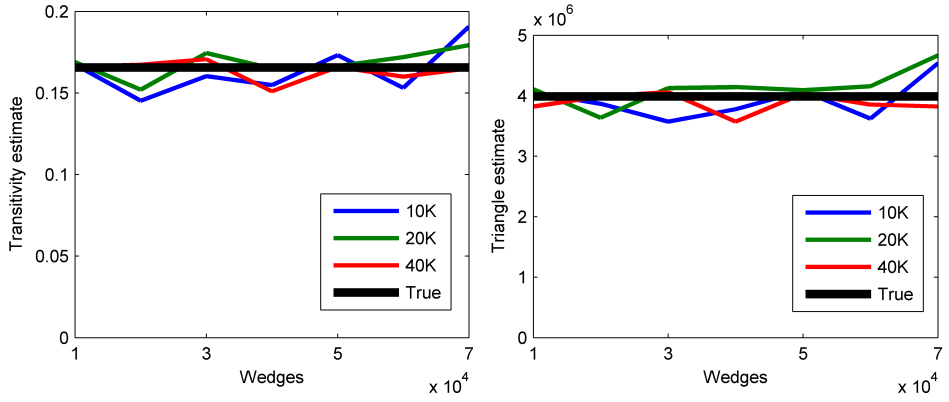
In Fig. 5, we compare runs of Streaming-Triangles with their algorithm for our test graphs. For convenience, we just refer to their algorithm as "Buriol et al". The x-axis is the space used, and the y-axis gives the triangle estimate. For simplicity, we fix s_e = 20K and increase s_w in Streaming-Triangles. For Buriol et al, we count the storage of a single edge and vertex a single unit of space. (We ignore the extra two edges needed to complete the triangle, and simply count the number of samples used.) While the previous algorithm has estimates off by 50% or more, Streaming-Triangles is much more accurate over all of its runs. (Indeed, the figure is zoomed out so much that the statistical flucations of Streaming-Triangles are barely visible.) For amazon0505, the estimate give by Buriol et al is zero till 80K samples. We note that these results are consistent with those given in [10].

Runs on various graphs: We run Streaming-Triangles on a variety of graphs obtained from the SNAP database [41]. We simply set s_e as 20K and s_w as 20K for all our runs. Each graph is converted into a stream by taking a random ordering on the edges. In Fig. 6, we show our re-



(a) web-NotreDame: Transitivity (b) web-NotreDame: Triangles

Figure 3: web-NotreDame detailed runs: We fix the edge reservoir, s_e, to 10K, 20K, and 40K. Then we increase the wedge reservoir size, s_w, from 10K to 70K and track the output of STREAMING-TRIANGLES. We give plots for transitivity and triangle counts.



(a) amazon0505: Transitivity (b) amazon0505: Triangles

Figure 4: amazon0505 detailed runs: We fix the edge reservoir, s_e, to 10K, 20K, and 40K. Then we increase the wedge reservoir size, s_w, from 10K to 70K and track the output of STREAMING-TRIANGLES. We give plots for transitivity and triangle counts.

sults for estimating both κ and T. The absolute values are plotted for κ together with the true values. For triangles, we plot the relative error (so $|est - T|/T$, where est is the algorithm output) for each graph, since the true values can vary over orders of magnitude. Observe that the transitivity estimates are very accurate. The relative error for T is mostly below 8%, and often below 4%.

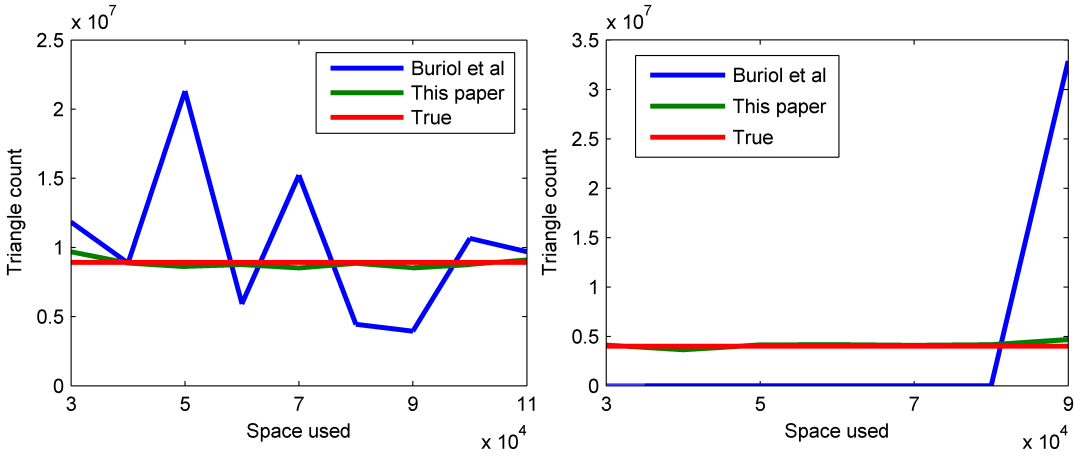
All the graphs listed have millions of edges, so our storage is always 2 orders of magnitude smaller than the graph. Most dramatically, we get accurate results on the `Orkut` social network, which has 220M edges. *The algorithm stores only 30K edges, a 0.0001-fraction of the graph.* Also observe the results on the `Flickr` and `Livejournal` graphs, which also run into tens of millions of edges.

Effect of stream ordering: It is not possible to check that our algorithm works for all orderings of the stream. So we generate a different orderings of the edges in web-NotreDame and run STREAMING-TRIANGLES on all the orderings. The results are given in Fig. 2. As before, we fix the edge and wedge reservoir to 20K. We discuss the different orderings below.

We first have two random orderings. Next, we generate a stream through a bfs as follows. We take a bfs tree from a random vertex and list out all edges in the tree. Then, we list the remaining edges in random order. Since the average degree of web-NotreDame is around 3, the tree has about one-third of the total edges. So this ordering is fairly different from a random ordering. Our fourth ordering involves taking a dfs from a random vertex and list out edges in order as seen by the dfs. Finally, we sort the vertices by degree and list all edges incident to a vertex (this is an incidence stream).

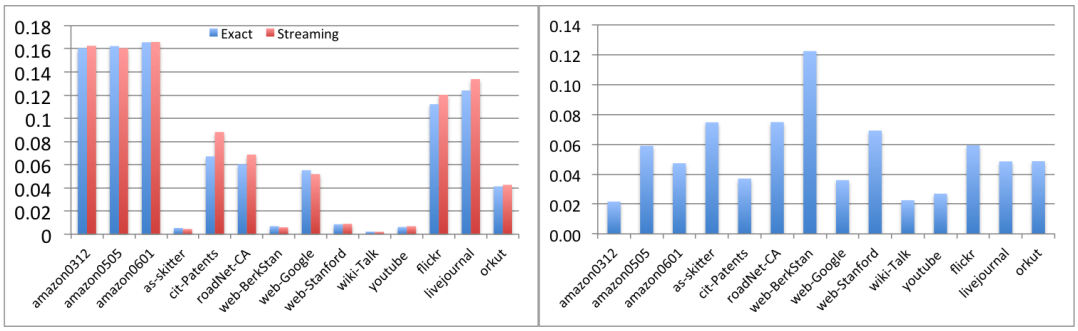
STREAMING-TRIANGLES performs reasonably on all these different orderings. There is little deviation in the transitivity values. There is somewhat more difference in the triangle numbers, but it never exceeds 10% relative error. This seems to be simply deviations in repeated runs of STREAMING-TRIANGLES.

Real-time tracking: A major benefit of STREAMING-TRIANGLES is that it can maintain a real-time estimate of κ_t and T_t. We take a real-world temporal graph, `cit-Patents`, which contains patent citation data over decades. The edges



(a) web-NotreDame (b) amazon0505

Figure 5: Comparison of our algorithm with Buriol et al [10] for amazon0505 and web-NotreDame. We use a 20K edge reservoir and variable wedge reservoir for our storage.



(a) Transitivity (b) Triangles

Figure 6: Output of a single run of Streaming-Triangles on a variety of real datasets with 20K edge reservoir and 20K wedge reservoir. The plot on the left gives the estimated transitivity values (labelled streaming) alongside their exact values. The plot on the right gives the relative error of Streaming-Triangles's estimate on triangles T. Observe that the relative error for T is mostly below 8%, and often below 4%.

are time stamped with the year of citation, and hence give a stream of edges. Using more storage of an edge reservoir of 40K and wedge reservoir of 15K, we accurately track these values over time (refer to Fig. 1). Note that this is still orders of magnitude smaller than the full size of the graph, which is 16M edges. The errors of our estimates are overall quite small. (The true values are only given for the year ends.)

8. REFERENCES

[1] K. J. Ahn, S. Guha, and A. McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *Principles of Database Systems*, pages 5–14, 2012.

[2] N. Alon, T. Kaufman, and M. Krivelevich. Testing triangle-freeness in general graphs. In *Proceedings of the 17th Annual Symposium on Discrete Algorithms (SODA)*, 2006.

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

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

[5] H. Avron. Counting triangles in large graphs using randomized matrix trace estimation. In *KDD workshon Large Scale Data Mining*, 2010.

[6] Z. Bar-Yossef, R. Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *Symposium of Discrete Algorith,s*, pages 623–632, 2002.

[7] 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)*, pages 16–24, 2008.

[8] 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 $\frac{7}{3}$.

Technical Report SAND2010-4474c, Sandia National Laboratories, 2011.

[9] 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)*, pages 1 –14, march 2007.

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

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

[12] 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*, pages 42–53, 2011.

[13] 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)*, pages 139–149, 1995.

[14] N. Chiba and T. Nishizeki. Arboricity and subgraph listing algorithms. *SIAM J. Comput.*, 14:210–223, 1985.

[15] S. Chu and J. Cheng. Triangle listing in massive networks and its applications. In *Knowledge Data and Discovery (KDD)*, pages 672–680, 2011.

[16] J. Cohen. Graph twiddling in a MapReduce world. *Computing in Science & Engineering*, 11:29–41, 2009.

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

[18] 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)*, 99(9):5825–5829, 2002.

[19] W. Feller. *An Introduction to probability theory and applications: Vol I.* John Wiley and Sons, 3rd edition, 1968.

[20] M. Jha, C. Seshadhri, and A. Pinar. A space efficient streaming algorithm for triangle counting using the birthday paradox. Technical Report 1212.2264, Arxiv, 2012.

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

[22] 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)*, pages 598–609, 2012.

[23] M. N. Kolountzakis, G. L. Miller, R. Peng, and C. Tsourakakis. Efficient triangle counting in large graphs via degree-based vertex partitioning. In *WAW'10*, 2010.

[24] M. Latapy. Main-memory triangle computations for very large (sparse (power-law)) graphs. *Theoretical Computer Science*, 407:458–473, 2008.

[25] 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.

[26] R. Pagh and C. Tsourakakis. Colorful triangle counting and a mapreduce implementation. *Information Processing Letters*, 112:277–281, 2012.

[27] T. Plantenga. Inexact subgraph isomorphism in mapreduce. *Journal of Parallel and Distributed Computing*, (0), 2012.

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

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

[30] T. Schank and D. Wagner. Finding, counting and listing all triangles in large graphs, an experimental study. In *Experimental and Efficient Algorithms*, pages 606–609. Springer Berlin / Heidelberg, 2005.

[31] C. Seshadhri, A. Pinar, and T. G. Kolda. Fast triangle counting through wedge sampling. In *Proceedings of the SIAM Conference on Data Mining*, 2013.

[32] S. Suri and S. Vassilvitskii. Counting triangles and the curse of the last reducer. In *World Wide Web (WWW)*, pages 607–614, 2011.

[33] C. Tsourakakis. Fast counting of triangles in large real networks without counting: Algorithms and laws. In *International Conference on Data Mining (ICDM)*, pages 608–617, 2008.

[34] C. Tsourakakis, P. Drineas, E. Michelakis, I. Koutis, and C. Faloutsos. Spectral counting of triangles in power-law networks via element-wise sparsification. In *ASONAM'09*, pages 66–71, 2009.

[35] C. Tsourakakis, M. N. Kolountzakis, and G. Miller. Triangle sparsifiers. *J. Graph Algorithms and Applications*, 15:703–726, 2011.

[36] 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)*, pages 837–846, 2009.

[37] J. Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)*, 11(1):37–57, 1985.

[38] S. Wasserman and K. Faust. *Social Network Analysis: Methods and Applications*. Cambridge University Press, 1994.

[39] 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.

[40] J.-H. Yoon and S.-R. Kim. Improved sampling for triangle counting with MapReduce. In *Convergence and Hybrid Information Technology*, volume 6935, pages 685–689. 2011.

[41] Stanford Network Analysis Project (SNAP). Available at http://snap.stanford.edu/.