Counting Triangles in Large Graphs using Randomized Matrix Trace Estimation

Haim Avron*
School of Computer Science
Tel-Aviv University
haima@post.tau.ac.il

ABSTRACT

Triangle counting is an important problem in graph mining with several real-world applications. Interesting metrics, such as the clustering coefficient and the transitivity ratio, involve computing the number of triangles. Furthermore, several interesting graph mining applications rely on computing the number of triangles in a large-scale graph. However, exact triangle counting is expensive and memory consuming, and current approximation algorithms are unsatisfactory and not practical for very large-scale graphs. In this paper we present a new highly-parallel randomized algorithm for approximating the number of triangles in an undirected graph. Our algorithm uses a well-known relation between the number of triangles and the trace of the cubed adjacency matrix. A Monte-Carlo simulation is used to estimate this quantity. Each sample requires $O(|E|)$ time and $O(\epsilon^{-2}\log(1/\delta)\rho(G)^2)$ samples are required to guarantee an (ϵ,δ)-approximation, where $\rho(G)$ is a measure of the triangle sparsity of G ($\rho(G)$ is not necessarily small). Our algorithm requires only $O(|V|)$ space in order to work efficiently. We present experiments that demonstrate that in practice usually only $O(\log^2|V|)$ samples are required to get good approximations for graphs frequently encountered in data-mining tasks, and that our algorithm is competitive with state-of-the-art approximate triangle counting methods both in terms of accuracy and in terms of running-time. The use of Monte-Carlo simulation support parallelization well: our algorithm is embarrassingly parallel with a critical path of only $O(|E|)$, achievable on as few as $O(\log^2|V|)$ processors.

Categories and Subject Descriptors

F.2.0 [**Analysis of Algorithms and Problem Complexity**]: General; G.2.1 [**Combinatorics**]: Counting problems

General Terms

Algorithms, Experimentation

Keywords

Graphs, Triangles

*This research was supported in part by an IBM Faculty Partnership Award and by grant 1045/09 from the Israel Science Foundation (founded by the Israel Academy of Sciences and Humanities).

KDD-LDMTA'10, July 25, 2010, Washington, DC, USA.

1. INTRODUCTION

The use of graphs to model many real-world phenomena (like the World Wide Web, social networks and protein interaction networks) gives rise to many different graph-mining tasks. A very common graph-mining task is counting the number of triangles in the graph (i.e., the number of connected vertex triplets), because the number of triangles is closely connected with estimating the clustering coefficients and the transitivity ratio of the graph. Counting the number of triangles has other applications. In explorations of networks, triangles are a frequently used network statistic in the exponential random graph model [14, 19] and naturally appear in models of real-world network evolution [17]. Triangles have been used in several applications such as spam detection [7] and uncovering the hidden thematic layers in the web [12]. In the context of social networks triangles have a natural interpretation: friends of friends tend to be friends [20], and this can be used in link recommendation/friend suggestion [24].

However, counting triangles in large-scale graphs, with millions and billions of edges, is expensive. It is not practical to count triangles with a single machine, and parallel algorithms are required. Furthermore, the graph may be too large to hold in memory and streaming algorithms are needed (the total number of indexed nodes in the Web is in the order of 10^{10}, and the typical number of links per Web page is between 20 and 30 [7]).

In this paper we present TRACETRIANGLE, a new embarrassingly-parallel randomized algorithm for approximating the number of triangles. TRACETRIANGLE is based on a numerical approximation of the trace (sum of diagonal value) of A^3, where A is the adjacency matrix of the graph, using a Monte-Carlo simulation. It is a well-known fact that the number of triangles is equal to $\text{trace}(A^3)/6$. Each Monte-Carlo sample requires $O(|E|)$ operations. We present three variants, based with a different sampling space. The first variant is theoretically the best, requiring $O(\epsilon^{-2}\log(1/\delta)\rho(G)^2)$ to guarantee an (ϵ,δ)-approximation, where $\rho(G)$ is a measure of the triangle sparsity ($\rho(G)$ can be very large, but it seems that often it is not too large). Weaker bounds are be established for other variants, but these variants have advantages in

Table 1: Notations and symbols used throughout the paper.

$G = (V, E)$	Undirected graph, no loops.
$n = \|V\|$	# of vertices.
$\triangle(G)$	# of triangles in G.
A	Adjacency matrix of G.
$\lambda_1, \ldots, \lambda_n$	Eigenvalues of A.
trace(M)	Sum of diagonal elements, trace(M) $= \Sigma_{i=1}^{n} M_{ii} = \Sigma_{i=1}^{n} \lambda_i$

terms of implementation. Only $O(|V|)$ space is required in addition to the space required for storing the graph. The algorithm can be implemented efficiently when the graph is kept in secondary storage, therefore effectively only $O(|V|)$ space is needed. TRACETRIANGLE is embarrassingly parallel: most of the work can be done in parallel without any synchronization. The critical path is $O(|E|)$ (the work required for a single sample).

We report an experimental evaluation of TRACETRIANGLE. We show that in practice usually only $O(\log^2 |V|)$ samples are required to get good approximation for graphs frequently encountered in data-mining applications. We compare TRACETRIANGLE to the state-of-the-art method EIGENTRIANGLE [23] and show that TRACETRIANGLE is competitive both in terms of accuracy and in terms of running-time.

2. RELATED WORK

2.1 Triangle counting algorithms

A typical strategy for counting triangles is to list all triangles. The most widely-used listing methods are the NODEITERATOR and the EDGEITERATOR algorithms. The NODEITERATOR algorithm considers each one of the nodes and examines which pairs of its neighbors are connected. The EDGEITERATOR algorithm computes for each edge the number of triangles that contain it. Asymptotically, both methods have the same time complexity: $\Theta(\sum_{v \in V} \deg(v)^2)$, where $\deg(v)$ is the degree of v. NODEITERATOR and EDGEITERATOR's have two major drawbacks when it comes to counting triangles in very large-scale graphs. First, they do not parallelize well. It is hard to load-balance work and space requirements between compute units. We are not aware of any paper presenting an efficient parallel implementation of NODEITERATOR and EDGEITERATOR. Secondly, both algorithm traverse the graph randomly, so an efficient implementation is required to store the graph in main memory. In contrast, our method traverses the graph sequentially so it can be kept in secondary storage without a big penalty.

The theoretically most efficient counting algorithm is due to Alon, Yuster and Zwick [4]. It is based on fast matrix multiplication, and runs in $\Theta(|E|^{\frac{2\omega}{\omega+1}})$ where ω is the fast matrix multiplication exponent. Thus, the Alon et al. algorithm currently runs in $O(|E|^{1.41})$ time. Space complexity is $\Theta(|V|^2)$ which is prohibitive to for large graphs.

In many cases an exact count of triangles is not crucial, and an high quality approximation will suffice. The goal such an algorithm is to produce a relative ϵ-approximation of the number of triangles with high probability. Most of the approximate triangle counting algorithms have been developed in the streaming model, i.e. these algorithms make only a constant number of passes over the input, usually by employing some form of Monte-Carlo simulation.

The simplest method uses naive sampling: three random nodes are tested whether they form a triangle [21]. The total number of triangles is estimated as the fraction of the triples that formed a triangle. For a fixed ϵ and failure probability the number of samples required to find an ϵ-approximation is $\Theta(\frac{T_0+T_1+T_2}{T_3})$ where T_i = #of triples with i edges. For graphs with $o(|V|^2)$ triangles this approach is not suitable. Unfortunately, this is the typical case when dealing with real-world networks.

A much more subtle approach was presented in [6]. The authors reduce the problem of triangle counting efficiently to estimating moments for a stream of node triples. They then use algorithm presented by Alon-Matias-Szegedy [3] to proceed. Along the same lines, Buriol et al. [8] proposed two space-bounded sampling algorithms to estimate the number of triangles. Again, the underlying sampling procedures are simple. For the case of the edge stream representation, they sample randomly an edge and a node in the stream and check if they form a triangle. Their algorithms are the state-of-the-art Monte-Carlo algorithms to the best of our knowledge, but they still require the graph to be fairly dense in order to get a good approximation.

Recently, a new triangle counting accelerator has been suggested by Tsourakakis et al. [25]. The algorithm randomly throws out a fraction of the edges, and then counts the number of triangles in the remaining graph (using any other triangle counting algorithm). An approximation for the number of triangles in the full graph can be computed from the exact count of triangles in the smaller graph. More theoretical analysis is provided in [9]. The aim of their algorithm is to enable the acceleration of a core triangle counting algorithm: after the initial sparsification the actual number of triangles is computed using some other triangle counting algorithm. The triangle sparsification, can, of course, be parallelized quite easily. But in order to fully parallelize a triangle counting code the core algorithm must be parallel as well. Tsourakakis et al. do not suggest any new way to parallelize the core algorithm. Furthermore, they not solve the issue of memory load. The number of bytes held in memory drops, but only by a constant factor. It is still the case that the entire (sparsified) graph must be kept in main memory for an efficient implementation.

2.2 Spectral counting

Tsourakakis recently presented a method based on computing eigenvalues [23]. It is based on the following observation [10].

LEMMA 1. *The total number of triangles $\triangle(G)$ in an undirected graph is equal to the trace of A^3 and to the sum of the cubes of the eigenvalues of the adjacency matrix divided by six, i.e.:*

$$\triangle(G) = \frac{1}{6} \operatorname{trace}(A^3) = \frac{1}{6} \sum_{i=1}^{n} \lambda_i^3 .$$

Tsourakakis observed that the absolute values of the eigenvalues tend to be skewed, typically following a power law [13, 18, 11]. Therefore, the number of triangles can be well approximated using the largest magnitude eigenvalues. The algorithm, EIGENTRIANGLE, uses a Lanczos iteration to find largest magnitude eigenvalues, and if implemented using

state-of-the-art software like ARPACK it is very fast. The method main drawbacks are: it is hard to determine how many eigenvalues need to be computed, there are no approximation guarantees, nor are there any guarantees on the number of steps the algorithm will do. We also note that there is a data-dependency between Lanczos iterations. The algorithm can be parallelized by parallelizing the matrix-vector multiplication, an operation whose parallelism is usually modeled as the ability to partition a graph. For some sparse matrices only very mediocre parallelism can be achieved. For some treatment on the subject see [15]. We do not know if matrices frequently encountered in graph mining tasks exhibit this problem, and we are unaware of any study on these issues. Here it should be noted that the use of ScaLAPACK is not an option as it is targeted for dense matrices, while the matrices involved in triangle counting are sparse.

In contrast, in our method there is no data-dependency between matrix-vector multiplication, so our algorithm can benefit from both the parallelism of the matrix-vector operation, and the parallelism inherent in the use of independent Monte-Carlo samples.

2.3 Numerical trace estimation

Computing the trace of an explicit matrix is a computationally cheap operation. However, in some cases the matrix can only be given implicitly as an operation on vectors. Counting triangles falls into that category: for large real networks it is impossible to actually form A^3.

The standard Monte-Carlo simulation for estimating the trace of an implicit matrix is due to Hutchinson [16], who proves the following Lemma.

LEMMA 2. *Let A be an $n \times n$ symmetric matrix with* $\mathrm{trace}(A) \neq 0$. *Let z be a random vector whose entries are i.i.d Rademacher random variables (*$\Pr(z_i = \pm 1) = 1/2$*). $z^T Az$ is an unbiased estimator of* $\mathrm{trace}(A)$ *i.e.,* $\mathrm{E}(z^T Az) = \mathrm{trace}(A)$ *and* $\mathrm{Var}(z^T Az) = 2\left(\|A\|_F^2 - \sum_{i=1}^n A_{ii}^2\right)$.

It is easy to see that for a general matrix Hutchinson's method can be ineffective because the variance can be arbitrarily large. Even for a symmetric positive definite the variance can be large: the variance for the matrix of all 1's, which is symmetric semi-definite, is $n^2 - n$, whereas the trace is only n. Such a large variance precludes the use of Chebyshev's inequality to establish a bound on the number of iterations required to obtain a given relative error in the trace. For such a bound to hold, the variance must be $o(\mathrm{trace}(A)^2)$. It is hard to use Lemma 2 to give rigorous bounds on the number of samples/matrix multiplications, and this difficulty carries over to applications of this method.

Recently, rigorous bounds have been discovered using Chernoff-type analysis [5]. If the entries of z are i.i.d standard normal variables then $O(\epsilon^{-2}\log(1/\delta))$ samples are required for an (ϵ, δ)-approximation for a positive definite matrix. For i.i.d Rademacher random variables (i.e., Hutchinson's method) $O(\epsilon^{-2}\log(n/\delta))$ samples are required for a positive definite matrix. The analyses in this paper are based on these results.

3. ALGORITHM

We now describe the first variant of TraceTriangle and state its properties. This is the variant with the best bounds. The other two variants, which describe in Sections 4 and 5, will enjoy some advantages but have weaker bounds. In order to shorten the expressions in the complexity analysis we assume that $|E| \geq |V|$, although it is easy to generalize the results for other cases.

Algorithm 1 TraceTriangle$_{\mathrm{N}}$

$\triangle$=**TraceTriangle**$_{\mathrm{N}}$(G, undirected graph with n nodes)
▷ parameter: γ.

1. Form the adjacency matrix $A \in \mathbb{R}^{n \times n}$.
2. $M = \lceil \gamma \ln^2 n \rceil$
3. Do M times, possibly in parallel
4. Form the vector $x = (x_k)$, where $x_i \sim N(0,1)$ are i.i.d random variables.
5. $y \longleftarrow Ax$
6. $T_i \longleftarrow (y^T Ay)/6$,
   where i is the index of the do loop
7. end
8. $\triangle \longleftarrow \frac{1}{M}\sum_{i=1}^M T_i$

We propose Algorithm 1, which we call TraceTriangle$_{\mathrm{N}}$. The subscript (N) denotes that this variant uses standard normal variables in line 4. TraceTriangle$_{\mathrm{N}}$ uses Monte-Carlo simulation to estimate the trace of A^3. Instead of using vectors composed of Rademacher random variables standard normal variables are used, since they allow us to prove better bounds. The number of Monte-Carlo samples is set to $O(\log^2 |V|)$. For a large class of graphs this is sufficient, although it might not be sufficient for graphs with very few triangles.

The following definition and theorem summarizes the main properties of TraceTriangle$_{\mathrm{N}}$. The definition describes a measure of how sparse triangles are in the graph.

DEFINITION 1. *The* eigenvalue triangle sparsity *of a graph is G is*

$$\rho(G) = \frac{\sum_{i=1}^n \left|\lambda_i^3\right|}{6\triangle(G)}.$$

If there are no triangles in the graph then $\rho(G) = \infty$.

THEOREM 1. *Given a fixed approximation ratio $\epsilon > 0$, fixed probability of failure $\delta > 0$ and fixed sparsity growth ratio $C \geq 1$ there exists a $\gamma = f(\epsilon, \delta, C)$ such that* TraceTriangle$_{\mathrm{N}}$ *with γ as the parameter in line 2 on undirected graphs for which $\rho(G) \leq C \log |V|$ will return Δ such that*

$$(1-\epsilon)\triangle(G) \leq \triangle \leq (1+\epsilon)\triangle(G)$$

with probability of at least $1-\delta$. Furthermore, $\gamma = O(\epsilon^{-2}C^2\log(1/\delta))$. The running time of the algorithm is $\Theta(|E|\log^2|V|)$ (for a fixed ϵ, δ and C) and it requires only $\Theta(|V|)$ space in addition to the space required for keeping the matrix (which is $O(|E|)$).

TraceTriangle$_{\mathrm{N}}$ guarantees an (ϵ, δ)-approximation only for graphs with logarithmically bounded triangle sparsity (i.e., $\rho(G) = O(\log |V|)$). The number of samples could have been decoupled from an a-priori bound on the sparsity by setting $M = O(\epsilon^{-2}\rho(G)^2\log(1/\delta))$, but the quantity $\rho(G)$ is unknown and probably impossible to approximate as it involves two unknown quantities: the number of triangles and $\sum_{i=1}^n \left|\lambda_i^3\right|$, so instead we decided to limit the number of samples and characterize the class of graphs for which our algorithms finds an (ϵ, δ) approximation. Furthermore, even if $\rho(G)$ was known in advance it would be impractical

to use the explicit formula for γ. Reasonable values like $\epsilon = 0.05$ and $\delta = 0.05$ will result in a huge sample size. In practice, as the experiments shown in Section 6 show, we noticed that $O(\log^2 |V|)$ usually gives good approximations for graphs encountered in data-mining applications, so this is the sample size we used, thereby bounding running time to $\Theta(|E| \log^2 |V|)$.

We do, however, prove a simple, yet not fully satisfactory, bound on $\sum_{i=1}^{n} |\lambda_i^3|$.

PROPOSITION 1. *Let A be the adjacency matrix of an undirected graph $G = (V, E)$. Let $\lambda_1, \ldots, \lambda_n$ be A's eigenvalues. The following hold*

$$\sum_{i=1}^{n} |\lambda_i^3| \leq \sqrt{8} |E|^{3/2} .$$

PROOF. Let x be the vector composed of the absolute eigenvalues. Since $\|x\|_2^2 = \|A\|_F^2 = |E|$ and $\|x\|_3 \leq \|x\|_2$ we have

$$\sum_{i=1}^{n} |\lambda_i^3| = \|x\|_3^3 \leq \|x\|_2^3 = \sqrt{8} |E|^{3/2} .$$

□

Real-world graphs usually obey a power-law distribution of the degrees. This implies that $|E| \approx c|V|$ ([13]) which in turn implies that if $\triangle(G) = \Omega(|V|^{3/2} / \log |V|)$ our algorithm will achieve a (ϵ, δ)-approximation with a constant γ (depending on ϵ, δ and c). The relation between number of triangles and the number of vertices and edges is not clear. Sergi [22] analyzes the number of triangles in random graph models and in some cases it is $\Omega(|E|^{3/2})$ but sometimes it is less. Tsourakakis [23] experimentally shows that the number of triangles per vertex in real-life networks follows a power-law with respect to the vertex degree. His experiments suggest that for real-life networks $\triangle(G) \sim \sum_{v \in V} \deg(v)^{\beta}$ where $\beta \geq 1.5$. In any case this is already an improvement over older randomized counting algorithm which require $\triangle(G) = \Omega(|V|^2)$.

TRACETRIANGLE has a few other desirable properties. Unlike some approximation methods, like EIGENTRIANGLE, there is no threshold: the number of steps needed for completion is known in advance. The number of random bits required is only $O(|V| \log^2 |V|)$. It is embarrassingly parallel: each of the samples can be processed in parallel without any synchronization. The only step that requires synchronization is the summation in step 8. Even without parallelzing the implicit matrix multiplication the critical path is $O(|E|)$, which can be achieved with only $O(\log^2 |V|)$ processors. TRACETRIANGLE is also a streaming algorithm (i.e., it can be implemented with $O(1)$ passes over the graph), although as a streaming algorithm the space requirement is $O(|V| \log^2 |V|)$ which is very large. As a semi-streaming algorithm space requirement are $O(|V| \log |V|)$ and $O(\log |V|)$ passes are needed. $O(|V|)$ space is achievable with $O(\log^2 |V|)$ passes. For some graphs the space requirements may be large, but luckily TRACETRIANGLE is also embarrassingly parallel in its memory usage: with $\log^2 |V|$ independent machines (a cluster) each machine will use only $O(|V|)$ space. Connected to this issue, is the observation that TRACETRIANGLE traverses the graph sequentially, so the graph itself can be kept in secondary storage without paying a big penalty. This is especially important for very large-scale graphs that cannot be held in main memory.

We now present the analysis of the trace estimator. Theorem 1 readily follows from this analysis. The first lemma shows that by using Gaussian variables we still get an unbiased estimator. We omit the proof.

LEMMA 3. *Let A be an $n \times n$ symmetric matrix with $\operatorname{trace}(A) \neq 0$. Let z be a random vector whose entries are i.i.d standard normal random variables. $z^T A z$ is an unbiased estimator of $\operatorname{trace}(A)$ i.e., $\mathrm{E}(z^T A z) = \operatorname{trace}(A)$ and $\operatorname{Var}(z^T A z) = 2\|A\|_F^2$.*

The variance is not small enough to establish a bound on the error. Using Chernoff-type arguments we can bound the error.

DEFINITION 2. *A* Gaussian trace estimator *for a symmetric matrix $A \in \mathbb{R}^{n \times n}$ is*

$$G_M = \frac{1}{M} \sum_{i=1}^{M} z_i^T A z_i ,$$

where the z_i's are M independent random vectors whose entries are i.i.d standard normal variables.

NOTATION 1. *For a symmetric matrix $X \in \mathbb{R}^{n \times n}$ with eigenvalues $\lambda_1, \ldots, \lambda_n$ we denote*

$$\rho(X) = \frac{\sum_{i=1}^{n} |\lambda_i|}{\operatorname{trace}(X)} .$$

Note that if A is the adjacency matrix of a graph G then $\rho(G) = \rho(A^3)$.

LEMMA 4. *Let $\delta > 0$ be a failure probability and let $\epsilon > 0$ be a relative error. For $M \geq 20\epsilon^{-2}\rho(A)^2 \ln(4/\delta)$, the Gaussian trace estimator G_M of a symmetric matrix $A \in \mathbb{R}^{n \times n}$ satisfies*

$$\Pr\left(|G_M - \operatorname{trace}(A)| \leq \epsilon |\operatorname{trace}(A)|\right) \geq 1 - \delta .$$

PROOF. A is symmetric so it can be diagonalized. Let $\Lambda = U^T A U$ be the unitary diagonalization of A (its eigendecomposition), and define $y_i = U z_i$. Since U is a rotation matrix the entries of y_i are i.i.d Gaussian variables. Let $\lambda_1, \ldots \lambda_n$ be the eigenvalues of A, and assume that $\lambda_1, \ldots, \lambda_r$ are negative and $\lambda_{r+1}, \ldots, \lambda_n$ are positive. Notice that $G_M = \frac{1}{M} \sum_{i=1}^{M} \sum_{j=1}^{n} \lambda_j y_{ij}^2$ where $y_{ij}$is the j'th entry of y_i. Let us write $\tau_+ = \sum_{j=r+1}^{n} \lambda_j$ and $G_M^+ = \sum_{i=1}^{M} \sum_{j=r+1}^{n} \lambda_i y_{ij}^2$. We will show that if $M \geq 12\epsilon^{-2}\rho(A)^2 \ln(4/\delta)$ we have

$$\Pr\left(\left|G_M^+ - \tau_+\right| \leq \epsilon \tau_+ / \rho(A)\right) \geq 1 - \delta/2 .$$

The bound will be proved using a Chernoff-style argument. Since y_{ij} is standard normal then $\sum_{i=1}^{M} y_{ij}^2 \sim \chi^2(M)$. Therefore, the moment generating function of $Z = M G_M^+$ is

$$\begin{aligned} m_Z(t) &= \mathrm{E}(\exp(tZ)) \\ &= \prod_{i=r+1}^{n} (1 - 2\lambda_i t)^{-M/2} \\ &= (1 - 2\tau_+ t + h(t))^{-M/2} \end{aligned} \tag{1}$$

where

$$h(t) = \sum_{s=2}^{n} (-2)^s t^s \sum_{\substack{S \subseteq \Lambda_+(A) \\ |S| = s}} \prod_{x \in S} x$$

as long as $|\lambda_i t| \leq \frac{1}{2}$ ($\Lambda_+(A)$ is the set of positive eigenvalues of A).

It is easy to see if $x_1, \ldots, x_n$ is a set of non-negative real numbers. For all $i = 1, \ldots, n$ we have

$$\sum_{\substack{S \subseteq [n] \\ |S| = i}} \prod_{j \in S} x_j \leq \left(\sum_{i=1}^{n} x_i \right)^i .$$

Therefore, we can bound

$$|h(t)| \leq \sum_{j=2}^{n} (2\tau_+ t)^j .$$

Denote $\epsilon_0 = \epsilon/\rho(A)$. Set $t = \epsilon_0/4\tau_+(1+\epsilon_0/2)$. For $i = r+1, \ldots, n$ we have $\lambda_i t \leq \frac{1}{2}$ so (1) is the correct formula for $m_Z(t)$. We have

$$\begin{aligned} |h(t)| &\leq \sum_{j=2}^{n} \left(\frac{\epsilon_0}{2(1+\epsilon_0/2)} \right)^j \\ &\leq \frac{\epsilon_0^2}{4(1+\epsilon_0/2)} . \end{aligned}$$

Markov's inequality asserts that

$$\begin{aligned} \Pr\left(G_M^+ \geq \tau_+(1+\epsilon_0)\right) &= \Pr\left(Z \geq \tau_+ M(1+\epsilon_0)\right) . \\ &\leq m_Z(t) \exp(-\tau_+ M(1+\epsilon_0)t) \\ &\leq (1-\epsilon_0/2\,(1+\epsilon_0/2) \\ &\quad -\epsilon_0^2/4(1+\epsilon_0/2))^{-M/2} \\ &\quad \cdot \exp(-M \cdot \frac{\epsilon_0}{2} \cdot \frac{1+\epsilon_0}{1+\epsilon_0/2}) \\ &= \exp(-\frac{M}{2}(\ln(1-\epsilon_0/2) \\ &\quad +\frac{\epsilon_0}{2} \cdot \frac{1+\epsilon_0}{1+\epsilon_0/2})) \\ &\leq \exp(-M\epsilon_0^2/20) \end{aligned}$$

for $\epsilon_0 \leq 0.1$. We find that if
$M \geq 20\epsilon_0^{-2}\ln(4/\delta) = 20\epsilon^{-2}\rho(A)^2\ln(4/\delta)$ then
$\Pr\left(G_M^+ \leq \tau_+(1+\epsilon/\rho(A))\right) \leq \delta/4$. Using the same technique a lower bound can be shown.

Similarly we can show that if $\tau_- = \sum_{j=1}^{r} \lambda_j$ and $G_M^- = \sum_{i=1}^{M}\sum_{j=1}^{r} \lambda_i y_{ij}^2$ then

$$\Pr\left(\left|G_M^- - \tau_-\right| \leq \epsilon|\tau_-|/\rho(A)\right) \geq 1 - \delta/2 .$$

With probability of at least $1-\delta$ both events happen. Since $G_M = G_M^+ + G_M^-$ and $\mathrm{trace}(A) = \tau_+ + \tau_-$ if both event occur we have

$$\begin{aligned} |G_M - \mathrm{trace}(A)| &\leq |G_M^+ - \tau_+| + |G_M^- - \tau_-| \\ &\leq \frac{\epsilon}{\rho(A)}(\tau_+ + |\tau_-|) \\ &= \frac{\epsilon}{\rho(A)} \sum_{i=1}^{n} |\lambda_i| \\ &= \epsilon\,\mathrm{trace}(A) . \end{aligned}$$

□

Proof. (of Theorem 1) If we set $\gamma = 20\epsilon^{-2}C^2\ln(4/\delta)$ we find that $M = 20\epsilon^{-2}C^2\log^2|V|\ln(4/\delta)$. Since we assume that $\rho(G) \leq C\log|V|$ and $\rho(G) = \rho(A^3)$ we find that $M \geq 20\epsilon^{-2}\rho(G)^2\ln(4/\delta)$ so according to lemma 4 G_M is an (ϵ,δ) approximator. Algorithm 1 returns $G_M/6$ which is an (ϵ,δ) approximator to $\triangle(G)$.

The running time of every iteration (line 4-6) is $O(\#nnz(A)) = O(|E|)$ and there are $O(\log^2|V|)$ iterations, so overall running time is $O(|E|\log^2|V|)$. Except for the memory needed to store the adjacency matrix three vectors have to be kept in memory (x, y and Ay) so $O(|V|)$ space is need in addition to the space required to store the matrix. □

4. USING RADEMACHER VARIABLES

We now propose another variant of TraceTriangle, which is called TraceTriangle$_R$. In this variant we replace the standard normal distribution with the Rademacher distribution. That is, in line 4 of Algorithm 1 x_i is ± 1 with equal probability.

There are two reasons why this change makes sense. By using vectors composed of i.i.d Rademacher variables we are using the Hutchinson estimator which has a smaller variance then the Gaussian estimator (compare lemma 2 to lemma 3). The second reason is related to implementation. It is more expensive to generate samples from a standard normal distribution then it is to generate samples from the Rademacher distribution. Furthermore, in some settings, for example a database setting, implementing TraceTriangle$_N$ can be non-trivial, while working with ± 1 is much easier [1]. The following lemma analyzes convergence of Hutchinson's estimator. We omit the proof, and refer the reader to [5] for analysis of the positive definite case.

Definition 3. *An* Hutchinson trace estimator *for a symmetric matrix $A \in \mathbb{R}^{n\times n}$ is*

$$H_M = \frac{n}{M}\sum_{i=1}^{M} z_i^T A z_i ,$$

where the z_i's are M independent random vectors whose entries are i.i.d Rademacher random variables

Lemma 5. *Let $\delta > 0$ be a failure probability and let $\epsilon > 0$ be a relative error For $M \geq 6\epsilon^{-2}\rho(A)^2\ln(2\,\mathrm{rank}(A)/\delta)$, the Hutchinson trace estimator H_M of a symmetric matrix $A \in \mathbb{R}^{n\times n}$ satisfies*

$$\Pr\left(|H_M - \mathrm{trace}(A)| \leq \epsilon\,|\mathrm{trace}(A)|\right) \geq 1 - \delta .$$

Unfortunately, our current convergence bounds on Huthchinson's estimator introduce a new factor of $O(\log n)$ to the number of samples required. The experiments reported in Section 6 show that in practice Hutchinson's estimator is comparable to using a Gaussian estimator when the same number of samples is used (this may indicate that the bounds are weak), so we do not actually increase the number of samples.

5. REDUCING RANDOMNESS

Our third variant, which is called TraceTriangle$_M$, is designed to reduce the amount of randomness used by the algorithm. Although reducing randomness has some theoretical interest, in our case there is also a practical interest. The previous variants of TraceTriangle required $\Theta(|V|)$ random bits per sample. This is too many bits to generate and store before the Monte-Carlo tests. Therefore,

in a parallel implementation all processors will need access to the pseudo-random number generator, incurring synchronization costs. TRACETRIANGLE$_M$ uses only $O(\log |V|)$ random bits per sample, so all random bits can be generated in advance. A preprocessing step required an additional $O(|V|)$ random bits, so the overall cost is $O(|V| + M \cdot \log |V|)$. This is obviously less than $O(M \cdot |V|)$ required by the previous variants of TRACETRIANGLE. The reduction in randomness does not come without a price, at least from a theoretical standpoint: an additional factor of $O(\log^2 |V|)$ samples is required. The experiments reported in Section 6 show that the estimator used by TRACETRIANGLE$_M$ is comparable to using a Gaussian estimator when the same number of samples is used (this may indicate that the bounds are weak), so we do not actually increase the number of samples.

Our algorithm and its analysis are based on the following two ideas. The first idea is to restrict the sample vectors to a small sample space ($O(|V|)$ instead of $O(2^{|V|})$). More specifically we use unit vectors e_j as sample vectors. By reducing the size of the sample space to $O(|V|)$ we are able to form a sample with only $O(\log |V|)$ bits. The quadratic forms $e_j^T A e_j$ are simply the diagonal elements A_{jj}, so the expectation of $e_j^T A e_j$ is clearly $\mathrm{trace}(A)/n$. In contrast to previous variants, these quadratic forms do not depend in any way on the off-diagonal elements of A, only on the diagonal elements. The next lemma formalizes these observations. We omit the proof.

LEMMA 6. *Let A be an $n \times n$ symmetric matrix. Let j be a uniform random integer between 1 and n, and let $z = e_j$ be the j'th unit vector. $nz^T Az$ is an unbiased estimator of $\mathrm{trace}(A)$ i.e., $\mathrm{E}(nz^T Az) = \mathrm{trace}(A)$ and $\mathrm{Var}(nz^T Az) = n\sum_{i=1}^n A_{ii}^2 - \mathrm{trace}^2(A)$.*

The second idea is to not compute the trace of A^3, but the trace of $\mathcal{F}A^3\mathcal{F}^T$ where $\mathcal{F}$ is a unitary matrix. It is based on a simple consequence of lemma 1: we are free to manipulate the adjacency matrix using unitary transformations since they preserve eigenvalues. More specifically for any unitary matrix $\mathcal{F}$ we have $\triangle(G) = \frac{1}{6}\mathrm{trace}(\mathcal{F}A^3\mathcal{F}^T)$.

We construct $\mathcal{F}$ using a randomized algorithm that guarantees with high probability a relatively flat distribution of the diagonal elements of $\mathcal{F}A^3\mathcal{F}^T$. More precisely, we construct $\mathcal{F}$ in a way that attempts to flatten the distribution of all the elements of $\mathcal{F}A^3\mathcal{F}^T$, not just its diagonal elements. Our construction of $\mathcal{F}$ is the same as the one used in [2] to flatten entries of vectors.

DEFINITION 4. *A* random mixing matrix *is $\mathcal{F} = FD$, where F and D be n-by-n unitary matrices, and where D is diagonal matrix with diagonal entries that are i.i.d Rademacher random variables ($\Pr(D_{ii} = \pm 1) = 1/2$).*

The mixing matrix $\mathcal{F}$ is the product of two unitary matrices, a *seed* matrix F and a matrix D with ± 1 on the diagonal, so $\mathcal{F}$ is unitary. For $\mathcal{F}$ to be a good mixing matrix the value of $\eta = \max |F_{ij}|^2$ must be small. The following lemma shows this. We omit the proof, and refer the reader to [5] for analysis of the positive definite case.

DEFINITION 5. *A* random mixed trace estimator *for a symmetric matrix $A \in \mathbb{R}^{n\times n}$ is*

$$D_M = \frac{n}{M}\sum_{i=1}^{M} z_i^T \mathcal{F}A\mathcal{F}^T z_i \ ,$$

Algorithm 2 TRACETRIANGLE$_M$

$\triangle$=**TraceTriangle$_M$**(G, undirected graph with n nodes)
▷ parameter: γ (see Section 3).

1. Form the adjacency matrix $A \in \mathbb{R}^{n\times n}$.
2. Form D, an $n \times n$ diagonal matrix with equal probability ± 1 diagonal values.
3. $M = \lceil \gamma \ln^2 n \rceil$
4. Do M times, possibly in parallel
5. Assign to j a random integer drawn from $1, \ldots, n$ with uniform distribution.
6. Form the vector $x = (x_k)$ be defined by

$$x_k = \begin{cases} \sqrt{1/n} & j = 1 \\ \sqrt{2/n}\cos\left[\pi\left(k + \frac{1}{2}\right) j/n\right] & j \neq 1 \end{cases}$$

$(k = 0, \ldots, n-1)$

7. $y \longleftarrow ADx$
8. $T_i \longleftarrow n(y^T Ay)/6$, where i is the index of the do loop
9. end
10. $\triangle \longleftarrow \frac{1}{M}\sum_{i=1}^{M} T_i$

where the z_i's are M independent uniform random samples from $\{e_1, \ldots, e_n\}$, and $\mathcal{F}$ is a random mixing matrix with $\max |F_{ij}|^2 \leq 2/n$.

REMARK 1. *Notice that the matrix $\mathcal{F}$ is random, but is drawn only once. So in computing a single sample of D_M it the random mixing matrix $\mathcal{F}$ incurs a single cost of n random bits.*

LEMMA 7. *Let $\delta > 0$ be a failure probability and let $\epsilon > 0$ be a relative error. For $M \geq 8\epsilon^{-2}\rho(A)^2 \ln(4/\delta)\ln^2(4n^2/\delta)$, the random mixed trace estimator D_M of a symmetric matrix $A \in \mathbb{R}^{n\times n}$ satisfies*

$$\Pr\left(|D_M - \mathrm{trace}(A)| \leq \epsilon\,|\mathrm{trace}(A)|\right) \geq 1 - \delta\,.$$

The are a few possibilities for F. The minimal value of η for a unitary F is $1/n$. A normalized DFT matrix achieves the minimum, but applying it requires complex arithmetic. A normalized Hadamard matrix also achieves the minimum and its entries are real. However, Hadamard matrices do not exist for all dimensions, so they are more difficult to use (they require padding). The Discrete Cosine Transform (DCT) and the Discrete Hartley Transform (DHT), which are real, exist for any dimension, and can be applied quickly, but their η value is $2/n$, twice as large as that of the DFT and the Hadamard. All are valid choices. The decision should be based on the implementation cost of computing columns of F and applying DAD^T to them versus the value of η. In TRACETRIANGLE$_M$ we use the DCT matrix. The pseudocode for the algorithm is given in Algorithm 2. Line 6 is an explicit formula for the rows of the DCT matrix.

6. EXPERIMENTS

We experimented with all three variants of TRACETRIANGLE . This section reports the results of these experiments.

6.1 Experimental Setup

We compared a sequential implementation of TRACETRIANGLE, to EIGENTRIANGLE [23], the current state-of-the-art

sequential algorithm. Our algorithm was implemented in C. EigenTriangle is implemented using MATLAB's `eigs` function. `eigs` is, essentially, a MATLAB interface to ARPACK[1], which is implemented in Fortran77. EigenTriangle receives as a parameter the number of eigenvalues to compute. There is no a-priori way to determine how many eigenvalues are needed, nor is it efficient to compute some and then compute the rest. We therefore used a fixed value of 20 eigenvalues in our experiments.

Running times were measured on a machine with an Intel Core2 6400 CPU (we used only one core) running at 2.13 GHz with 4 GB of memory. The measured running times are wall-clock times that were measured using the `ftime` Linux system call.

The test graphs are listed in Table 2. All graphs except the AS-Oregon one are available from public data-sets. Each directed graph was converted into an undirected graph by ignoring the direction of the edges. Multiple edges and self-loops were removed.

6.2 Results

The goal of the first set of experiments was to compare the accuracy of different variants of TraceTriangle, and to compare them, in terms of accuracy, to EigenTriangle. For each graph we ran 100 experiments with various values of γ (100 runs for each value of γ). From the results we found the lowest value of γ that gives acceptable results. Table 3 shows the number of runs with error below 5% and the number of runs with error below 10% for all three variants of TraceTriangle. All variants of TraceTriangle act similarly. TraceTriangle$_{\mathrm{R}}$ is faster than other variants (experiments not reported) so we chose it as the variant to use in all other experiments.

Table 4 show additional statistics: the mean of the absolute error (in percents), the maximum absolute error (in percents), the number of runs with error below 5% and the number of runs with error below 10%.

Except from two graph (Web-Stanford and WikiNov2005) we were able to obtain results that are usually relatively accurate with $\gamma \leq 4$. A clear majority of the experiments resulted in an error of less than 10% and a majority resulted in error less than 5%. The maximum error is sometimes quite large ($\sim$ 20%) but this is not the typical case. For small web graphs (Web-Stanford and WikiNov2005) TraceTriangle seem to converge slowly. Notice that Wikipedia graphs are larger when the date is later, while better accuracy is achieved when the date progresses. This supports our choice to use a sample size of $O(\log^2 |V|)$: in seems that $\rho(G)$ does not grow too quickly.

The two rightmost columns of Table 4 show the deterministic error size of EigenTriangle. The column labeled "20 eigenvalues" shows the error of EigenTriangle with 20 eigenvalues. In the rightmost time we allow EigenTriangle to find as many eigenvalues as possible when it is given a running time that is about equal to TraceTriangle, and always larger than TraceTrinagle. The equal time configuration was chosen to avoid needlessly penalizing EigenTriangle, as one might claim that as the graph grows the number of eigenvalues that should be computed should grow as well. We notice that TraceTriangle is consistently more accurate than EigenTriangle.

[1]http://www.caam.rice.edu/software/ARPACK/

The results in Table 4 can be used to get some rule-of-thumb values for γ based on the graph type. For example, collaborations/citations graph tend to work with low values of γ (one or two), while web graphs, like the Wikipedia, require high values of γ. Arguably, the parameter that changes the most between different graphs is the value of $\rho(G)$. Different graph types have different typical values for $\rho(G)$. For collaboration/citations graphs $\gamma = 1 - 2$ seems adequate, for social networks $\gamma = 3$ and for large web graphs/communications networks $\gamma = 4$.

The second set of experiments we compares the running time of TraceTriangle$_{\mathrm{R}}$ to EigenTriangle. Both algorithms are deterministic in their running time, so only one run was necessary. Figure 2 shows the speedup (or slowdown) of TraceTriangle$_{\mathrm{R}}$ with respect to EigenTriangle. We see that for smaller graphs TraceTriangle$_{\mathrm{R}}$ is faster than EigenTriangle, sometimes with a large factor. For the Wikipedia graphs, which were the largest graphs in our test set, EigenTriangle was three times faster than TraceTriangle$_{\mathrm{R}}$. On the other hand, on the Wikipedia matrices EigenTriangle was not accurate. Furthermore, the ability to parallelize EigenTriangle is very limited, while TraceTriangle$_{\mathrm{R}}$ parallelizes well. We expect TraceTriangle$_{\mathrm{R}}$ to scale much better with respect to the number of compute cores, therefore eliminating any running time advantage of EigenTriangle.

Fixing the number of eigenvalues computed implies that the number of matrix-vector multiplication done by EigenTriangle grows very slowly. This explains why when the graph get larger EigenTriangle is faster than TraceTriangle. On the other hand as the graph get larger one can argue that more eigenvalue should be computed. In Figure 1 compare the accuracy-runtime tradeoff of TraceTriangle$_{\mathrm{R}}$ and EigenTriangle. We plot the accuracy of both algorithms for different running times. EigenTriangle is deterministic and its data points are based on a single runs with different #eigenvalues values. TraceTriangle is randomized, so for different values of γ we ran the algorithm 100 times and computed the mean absolute error. We see that TraceTriangle error is monotonically decreasing. EigenTriangle's error is not always monotonic, and is always higher than TraceTriangle's error. We note that currently there is no guideline for the number of eigenvalue in EigenTriangle (and this limits our ability to compare running time).

6.3 Doulion sparsification

Doulion [25] is an algorithm designed to accelerate triangle counting algorithms by randomly keeping only a small fraction of the edges. Doulion's authors suggest to use doulion with any triangle counting algorithm, so we tested doulion with TraceTriangle. The potential benefits for TraceTriangle are clear. If an edge is kept with probability p the expected number of edges after the sparsification is $p|E|$. Therefore, TraceTriangle is expected to speedup by roughly $1/p$.

The main problem in using doulion is that it changes the triangle sparsity. Every approximate counting algorithm must depend in some way on the triangle sparsity. In TraceTriangle the accuracy depends on $\rho(G)$ whose bound is expected to grow after sparsification. Other algorithms suffer from the same problem. Schank et al. method [21], for

Table 2: Graphs used for the experiments

Graph	Vertices	Edges	Source	Description
Arxiv-HEP-th	27,240	341,923	UF Sparse Matrix Collection[2], matrix 1502	Arxiv High Energy Physics paper citation network
CA-AstroPh	18,772	198,050	SNAP Network Data Repository[3], 'ca-AstroPh' matrix	Arxiv Astro Physics collaboration network
CA-GrQc	5,242	14,484	SNAP Network Data Repository, 'ca-GrQc' matrix	Arxiv GR-QC General Relativity and Quantum Cosmology collaboration network
Epinions	75,879	405,740	SNAP Network Data Repository, 'soc-Epinions1' matrix	Who-trusts-whom network of Epinions.com
Slashdot0811	77,360	469,180	SNAP Network Data Repository, 'soc-Slashdot0811' matrix	Slashdot social network from November 2008
wiki-vote	7,115	100,762	SNAP Network Data Repository, 'wiki-vote' matrix	Wikipedia who-votes-on-whom network
email-EuAll	265,214	364,481	SNAP Network Data Repository, 'email-EuAll' matrix	Email network from a EU research institution
AS Oregon	13,579	37,448	Courtesy of Charalampos Tsourakakis	AS Oregon
Web-Stanford	281,903	1,992,636	SNAP Network Data Repository, 'web-Stanford' matrix	Web graph of stanford.edu
WikiNov2005	1,634,989	18,549,197	UF Sparse Matrix Collection, matrix 1843	Wikipedia graph, Nov 11, 2005
WikiSep2006	2,983,494	35,048,116	UF Sparse Matrix Collection, matrix 1844	Wikipedia graph, Sep 25, 2006
WikiNov2006	3,148,440	37,043,458	UF Sparse Matrix Collection, matrix 1845	Wikipedia graph, Nov 4, 2006
WikiFeb2007	3,566,907	42,375,912	UF Sparse Matrix Collection, matrix 1846	Wikipedia graph, Feb 6, 2007

Table 3: Comparison between different variants of TraceTriangle, based on 100 runs. The "% err $< 5\%$" and "% err $< 10\%$" show the percent of cases for which the error was smaller than 5% and 10%.

		TraceTriangle$_N$		TraceTriangle$_R$		TraceTriangle$_M$	
Graph	γ	% err $< 5\%$	% err $< 10\%$	% err $< 5\%$	% err $< 10\%$	% err $< 5\%$	% err $< 10\%$
Arxiv-HEP-th	1	85%	100%	85%	100%	90%	100%
CA-AstroPh	1	98%	100%	98%	100%	100%	100%
CA-GrQc	2	70%	97%	77%	98%	72%	96%
Epinions	3	63%	95%	60%	94%	73%	96%
Slashdot0811	3	61%	91%	62%	95%	66%	95%
wiki-vote	3	51%	86%	52%	86%	52%	86%
email-EuAll	4	52%	81%	54%	85%	64%	93%
AS Oregon	4	45%	79%	39%	81%	53%	84%
Web-Stanford	3	36%	68%	40%	70%	36%	61%
WikiNov2005	4	34%	62%	34%	67%	33%	62%
WikiSep2006	4	58%	87%	69%	91%	57%	94%
WikiNov2006	4	65%	91%	58%	95%	62%	92%
WikiFeb2007	4	68%	97%	69%	99%	64%	95%

Table 4: Accuracy on various data sets, based on 100 runs of TraceTriangle$_R$ and a two (deterministic) runs of EigenTriangle: with 20 eigenvalues, and with running time equal to TraceTriangle. The "% err $< 5\%$" and "% err $< 10\%$" show the percent of cases for which the error was smaller than 5% and 10%.

Graph	γ	mean abs error	max abs error	% err $< 5\%$	% err $< 10\%$	EigenTriangle 20 eigenvalue	EigenTriangle $\gtrsim$ time
Arxiv-HEP-th	1	2.7%	6.7%	85%	100%	41%	75%
CA-AstroPh	1	1.6%	6.5%	98%	100%	47%	84%
CA-GrQc	2	3.2%	11.5%	77%	98%	12%	48%
Epinions	3	4.6%	12.3%	60%	94%	6.2%	9.6%
Slashdot0811	3	4.5%	16.0%	62%	95%	2.9%	3.1%
wiki-vote	3	5.6%	19.0%	52%	86%	1.9%	0.9%
email-EuAll	4	5.6%	22.4%	54%	85%	17%	16%
AS Oregon	4	6.5%	16.3%	39%	81%	1.5%	9.7%
Web-Stanford	3	7.9%	24.9%	40%	70%	28%	26%
WikiNov2005	4	8.7%	28.3%	34%	67%	49%	42%
WikiSep2006	4	4.1%	14.0%	69%	91%	54%	44%
WikiNov2006	4	4.4%	13.9%	58%	95%	54%	44%
WikiFeb2007	4	4.0%	11.3%	69%	99%	55%	48%

Figure 1: Closer inspection of the accuracy of TraceTriangle vs. EigenTriangle on WikiSep2006. We varied EigenTriangle's #eigenvalues parameter and TraceTriangle's γ parameter.

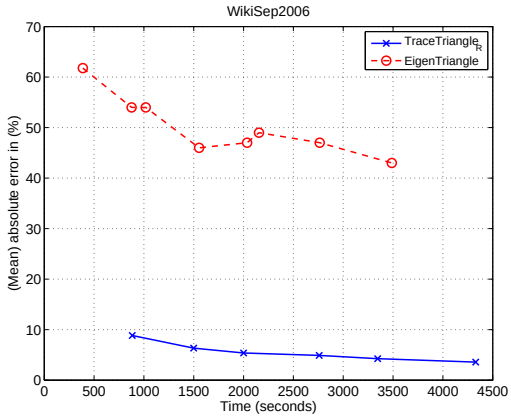

Figure 2: Speedup of TraceTriangle$_R$ with respect to EigenTriangle.

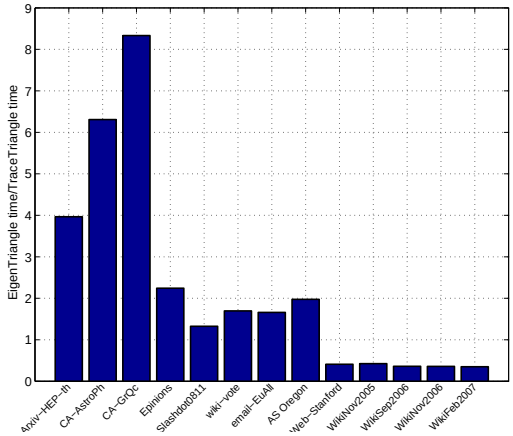

Table 5: Results involving TraceTriangle with doulion sparsification. Error is the mean of 10 runs for 10 different sparsifications.

Graph	p	mean error	speedup
CA-AstroPh	0.1	6.2%	3.5
Slashdot0811	0.2	12.5%	2.3
WikiNov2006	0.2	16.7%	4.2

example, requires $\Theta(\frac{T_0+T_1+T_2}{T_3})$ samples. After the sparsification the numerator grows and the denominator shrinks.

Table 5 shows the results for three matrices. In one case, triangle density was high enough after sparsification to still yield reasonable results. In other graphs triangle sparsity was too high and results are of low quality. Speedup is achieved at the cost of accuracy. We conclude that in order to effectively use graph sparsification in approximate triangle counting the sparsification algorithm must somehow keep the triangle sparsity close to the original. This is a feature that DOULION lacks. We currently are not aware of any such algorithm and leave this issue for future research.

7. CONCLUSIONS

We have presented an embarrassingly parallel algorithm for approximating the number of triangles in a graph. We showed that given sufficient triangle density our algorithm gives a good estimate to the number of triangles with high probability. We also showed that for real-life graphs encountered in graph-mining applications our algorithm approximates the number of triangles well. An interesting feature of our algorithm is the use of a randomized numerical linear algebra algorithm to solve a combinatorial problem. Our main contributions are:

- The new triangle counting algorithm and its theoretical analysis.
- Experiments on large graphs with several millions of vertices and edges.

Acknowledgments

It is a pleasure to thank Mark Tygert and Sivan Toledo for helpful comments and ideas, and Charalampos Tsourakakis for helpful discussions and for supplying his triangle counting codes.

8. REFERENCES

[1] Dimitris Achlioptas. Database-friendly random projections. In *PODS '01: Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 274–281, New York, NY, USA, 2001. ACM.

[2] Nir Ailon and Bernard Chazelle. Approximate nearest neighbors and the fast johnson-lindenstrauss transform. In *STOC '06: Proceedings of the thirty-eighth annual ACM Symposium on Theory of Computing*, pages 557–563, New York, NY, USA, 2006. ACM.

[3] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *STOC '96: Proceedings of the twenty-eighth annual ACM Symposium on Theory of Computing*, pages 20–29, New York, NY, USA, 1996. ACM.

[4] Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17:354–364, 1998.

[5] Haim Avron and Sivan Toledo. Randomized algorithms for estimating the trace of an implicit symmetric positive semi-definite matrix. Submitted for publication, 2010.

[6] Ziv Bar-Yossef, Ravi Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *SODA '02: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 623–632, Philadelphia, PA, USA, 2002. Society for Industrial and Applied Mathematics.

[7] Luca Becchetti, Paolo Boldi, Carlos Castillo, and Aristides Gionis. Efficient semi-streaming algorithms for local triangle counting in massive graphs. In *KDD '08: Proceeding of the 14th ACM SIGKDD international conference on Knowledge Discovery and Data mining*, pages 16–24, New York, NY, USA, 2008. ACM.

[8] Luciana S. Buriol, Gereon Frahling, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Christian Sohler. Counting triangles in data streams. In *PODS '06: Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 253–262, New York, NY, USA, 2006. ACM.

[9] Tsourakakis C., Kolountzakis M., and Miller G. L. Approximate triangle counting. Arxiv 0904.3761, 2009.

[10] Godsil C.D. and Royle G. *Algebraic Graph Theory*. Springer, 2001.

[11] Fan Chung, Linyuan Lu, and Van Vu. Eigenvalues of random power law graphs. *Annals of Combinatorics*, 7(1):21–33, June 2003.

[12] Jean-Pierre Eckmann and Elisha Moses. Curvature of co-links uncovers hidden thematic layers in the world wide web. *PNAS*, 99(9):5825–5829, April 2002.

[13] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. In *SIGCOMM*, pages 251–262, 1999.

[14] Ove Frank and David Strauss. Markov graphs. *Journal of the American Statistical Association*, 81(395):832–842, 1986.

[15] Bruce Hendrickson. Graph partitioning and parallel solvers: Has the emperor no clothes? (extended abstract). In *In Proc. Irregular'98*, pages 218–225. Springer-Verlag, 1998.

[16] M. F. Hutchinson. A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. *Communications in Statistics, Simulation and Computation*, 19(2):433–450, 1989.

[17] Jure Leskovec, Lars Backstrom, Ravi Kumar, and Andrew Tomkins. Microscopic evolution of social networks. In *KDD '08: Proceeding of the 14th ACM SIGKDD international conference on Knowledge Discovery and Data mining*, pages 462–470, New York, NY, USA, 2008. ACM.

[18] Milena Mihail and Christos H. Papadimitriou. On the eigenvalue power law. In *Proceedings of the 6th International Workshop on Randomization and Approximation Techniques (RANDOM)*, pages 254–262, London, UK, 2002. Springer-Verlag.

[19] Alessandro Rinaldo, Stephen E. Fienberg, , and Yi Zhou. On the geometry of discrete exponential families with application to exponential random graph models. *Electronic Journal of Statistics*, 3:446–484, 2009.

[20] Wasserman S. and Faust K. *Social Network Analysis: Methods and Applications*. Cambridge University Press, 2004.

[21] Thomas Schank and Dorothea Wagner. Approximating clustering-coefficient and transitivity. *Journal of Graph Algorithms and Applications*, 9(2):265–275, 2004.

[22] Danilo Sergi. Random graph model with power-law distributed triangle subgraphs. *Phys. Rev. E*, 72(2):025103, Aug 2005.

[23] Charalampos E. Tsourakakis. Fast counting of triangles in large real networks without counting: Algorithms and laws. *IEEE International Conference on Data Mining (ICDM 2008)*, 0:608–617, 2008.

[24] Charalampos E. Tsourakakis, Petros Drineas, Eirinaios Michelakis, Ioannis Koutis, and Christos Faloutsos. Spectral counting of triangles via element-wise sparsification and triangle-based link recommendation. To appear as a book chapter in *Advances in Social Networks Analysis and Mining*.

[25] Charalampos E. Tsourakakis, U. Kang, Gary L. Miller, and Christos Faloutsos. Doulion: counting triangles in massive graphs with a coin. In *KDD '09: Proceedings of the 15th ACM SIGKDD international conference on Knowledge Discovery and Data mining*, pages 837–846, New York, NY, USA, 2009. ACM.