DOULION: Counting Triangles in Massive Graphs with a Coin

Charalampos E. Tsourakakis, U Kang, Gary L. Miller, and Christos Faloutsos
SCS, Carnegie Mellon University
Pittsburgh, PA, USA
ctsourak@cs.cmu.edu, ukang@cs.cmu.edu, glmiller@cs.cmu.edu,
christos@cs.cmu.edu

ABSTRACT

Counting the number of triangles in a graph is a beautiful algorithmic problem which has gained importance over the last years due to its significant role in complex network analysis. Metrics frequently computed such as the clustering coefficient and the transitivity ratio involve the execution of a triangle counting algorithm. Furthermore, several interesting graph mining applications rely on computing the number of triangles in the graph of interest.

In this paper, we focus on the problem of counting triangles in a graph. We propose a practical method, out of which all triangle counting algorithms can potentially benefit. Using a straightforward triangle counting algorithm as a black box, we performed 166 experiments on real-world networks and on synthetic datasets as well, where we show that our method works with high accuracy, typically more than 99% and gives significant speedups, resulting in even ≈ 130 times faster performance.

Categories and Subject Descriptors: F.2.2 Analysis of Algorithms and Problem Complexity: Nonnumerical Algorithms and Problems

General Terms: Algorithms; Experimentation.

Keywords: Graphs, Triangles, Hadoop

1. INTRODUCTION

Abundant data nowadays are modeled as graphs: the World Wide Web, social networks (e.g. LinkedIn, Facebook, Flickr) , P2P networks, co-authorship networks, biological networks, computer networks and physical connections just to name a few. Nowadays, due to the recent technology explosion, graphs reaching the planetary scale are available to us for analysis [21]. Triangles play an important role in complex network analysis. For example in social networks, triangles is a well studied subgraph. In particular, two prominent theories according to which triangles are generated in social networks are the homophily and the transitivity. According to the former, people tend to choose friends that are similar to themselves, which is also known as "birds of a feather flock together" [22]. and according to the latter, people who have common friends tend to become friends themselves [31].

The significance of the existence of triangles in networks motivates the definition of metrics that quantify the triangle density in a graph. Two such metrics are the clustering coefficient and the transitivity ratio [24].

Furthermore, it has been shown that triangles can play a significant role in graph mining applications as well. Recently, in [6] it was shown that triangles can be used to detect spamming activity. Eckman and Moses in [13] showed how triangles can be used to uncover the hidden thematic structure of the web. Moreover, according to [5], triangle counting can benefit the query plan optimization in databases. For the aforementioned reasons fast triangle counting algorithms are of high practical value.

In this paper we propose a simple, practical, yet effective algorithm for counting triangles in graphs. Our algorithm DOULION can be used in any graph. In our experiments we focus on real-world networks that exhibit a skewed degree distribution and in Erdős-Rényi graphs ([7]). DOULION is not a competitor of other triangle-counting algorithms. It is rather a "friend" since it can be used as a first step before applying any triangle counting algorithm, streaming or not. We verify the effectiveness of our method in a wide range of experiments on real-world networks and provide a basic mathematical analysis of our algorithm and some connections to the spectral analysis of matrices.

In figure 1 we see the results of running DOULION on one snapshot of the Wikipedia Web graph. As we see, even when keeping 10% of the edges accuracy is almost the ideal 100%. For the range of the "edge-keeping" percentages that we used, 10% to 90% with a step of 10% we received speedups 113.1, 28.9, 12.8, 7.1, 4.5, 3.1 2.2, 1.6, 1.3 correspondingly. The mean accuracy is 99.7% and the standard deviation 0.0023. DOULION has the advantage of being "embarassingly" parallel as well, therefore allowing us to easily implement it in any parallel programming framework. For our purposes, we used HADOOP the open source implementation of MAPREDUCE [11].

The outline of the paper is as follows: Section 2 presents an overview of the related work and Section 3 the proposed algorithm. Section 4 shows the experimental results and we conclude in section 5.

2. BACKGROUND AND RELATED WORK

In this section, we present the related work on the problem of counting triangles in a graph and briefly give some information on the MAPREDUCE framework and HADOOP.

2.1 Triangle Counting algorithms

Let $G(V, E)$, n=$|V|$, m=$|E|$ be an undirected, unweighted, simple graph. A triangle is a three-node subgraph of G which is fully connected.

KDD'09, June 28–July 1, 2009, Paris, France.

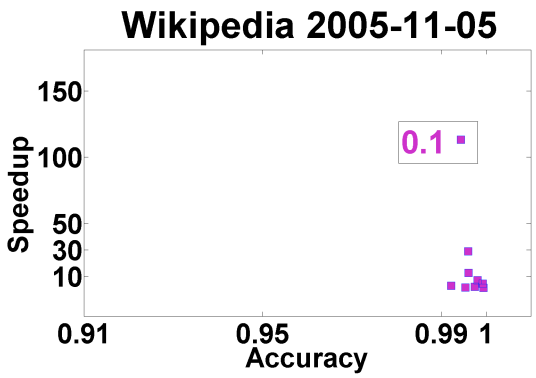


Figure 1: Speedup vs. Accuracy for the Wikipedia Graph snapshot on 2005 Nov. The graph has ≈ 1,7M nodes and 20M edges. As we see, even when keeping 10% of the edges of the initial graph accuracy is 99.5%. For p's ranging from 10% to 90% the mean accuracy is 99.7%, the accuracy standard deviation 0.0023 and the mean speedup 19.4.

Exact Counting Algorithms.

One obvious way to count the number of triangles in a graph is to enumerate all possible $\binom{n}{3}$ combinations of vertices and count how many of them are fully connected. This results in the naive algorithm with $O(n^3)$ time complexity.

A simple algorithm, known as NODEITERATOR, computes for each node its neighborhood and then sees how many edges exist among its neighbors. This algorithm runs asymptotically in $\sum_{v \in V(G)} \binom{d(v)}{2}$ time which by taking a simple union bound give an upper bound of $O(d_{max}^2 n)$, where d_{max} is the maximum degree in G. Another simple algorithm that works in a similar way is the EDGEITERATOR. Rather than checking each node at the time, EDGEITERATOR checks each edge $(u, v) \in E$ and computes the common neighbors of the nodes u and v. Asymptotically EDGEITERATOR runs in the the same time with the NODEITERATOR. This algorithm can be improved through a simple hashing argument so that it runs in $O(m^{\frac{3}{2}})$ [26]. This version of EDGEITERATOR is also called EDGEITERATOR-hashed. The $forward$ algorithm is another refinement of the EDGEITERATOR. The key idea of this algorithm is that there is no need to compare the full neighborhoods of two adjacent nodes. Finally the $compact - forward$ iterator ([20]) further improves the $forward$ algorithm. Itai and Rodeh in [16] gave an algorithm that finds a triangle if it exists in $O(m^{\frac{3}{2}})$. Their algorithm can easily be extended in a triangle counting algorithm with the same time complexity. Their algorithm relies on computing spanning trees of the graph G and removing edges while making sure that each triangle is listed exactly once. In [26] one can find the analysis and an extensive description of these algorithms.

The fastest methods for triangle counting in terms of time complexity are based on fast matrix multiplication. Alon et al. gave in [4] an algorithm of time complexity $O(m^{\frac{2\gamma}{\gamma+1}}) \subset O(m^{1.41})$ where at the time of this write-up γ is 2.37, the exponent of the state-of-the-art algorithm for matrix multiplication ([10]). Exact counting methods however may be slow, even not applicable when the size of the graph fairly large due to high memory requirements. In those cases an approximating algorithm is preferred in the cost of losing the exact number of triangles.

Streaming Algorithms.

The goal of streaming algorithms is to perform one or at most a constant number of passes over the graph stream (e.g. edges arriving one at a time $\{e_1, e_2, .., e_m\}$) and make provably accurate estimates of the number of triangles. Yossef et al. in their seminal paper [5] gave the first streaming algorithm for counting triangles. They first define all possible different triples that can show up and then reduce the problem of triangle counting to estimating moments for a stream of node triples. Then they use the Alon-Matias-Szegedy algorithms (also known as AMS algorithms) presented in the Gödel awarded work [3]. The space complexity of their algorithms depend on the structure of the graph, and specifically on the cardinalities of the sets of the different types of triples. In [17] three streaming algorithms were presented. Two of them use one pass over the graph stream and the third one three passes. The one-pass algorithms use again the AMS algorithms [3] and the later algorithm uses sampling to reduce the usage of space. The biased sampling is done according to the degree of the vertex chosen. In [8] two random sampling algorithms are proposed to estimate the number of triangles one for the edge stream representation and one for the incidence stream representation of the undirected graph of interest. The sampling procedures are simple. E.g., 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.

Semi-Streaming Algorithms.

Bechetti et al. presented in [6] a semi-streaming algorithm for computing triangles in a graph. Their model relaxes the strict constraint of constant number of passes to obtain an algorithm that performs $log(n)$ passes over the edge file. Their main idea relies on locality sensitivity hashing and the observation that the local triangle counting reduces to estimating the size of the intersection of two sets, namely the neighborhoods of two nodes connected by an edge. In [27] a spectral counting algorithm was introduced. The idea of this algorithm is to take advantage of the properties of the skewed spectra of power-law networks and make a fast approximation of the number of triangles based on a few, top eigenvalues. This algorithm can be viewed both as a semi-streaming algorithm in the sense that it performs a number of passes at worst $O(log(n))$ ([15]) over the non-zero elements of the adjacency matrix (edges) or even as a streaming algorithm by using a linear time algorithm for the SVD ([25]). The performance of the algorithm depends strongly on the spectrum of the graph of interest. Empirically the algorithm works well in many real-world graphs but has no guarantees, mainly due to the limited knowledge on the spectra of real-world graphs. We rather have theoretical knowledge on the few top eigenvalues ([23],[9]) or our knowledge is just empirical ([27],[14]).

2.2 MAPREDUCE

MAPREDUCE is a parallel distributed programming framework introduced in [12], which can process huge amounts of data in a massively parallel way using simple commodity machines. It is inspired by the functional programmming concepts of mapping and reducing. HADOOP - rougly speaking - is the open source implementation of MAPREDUCE. It is an emerging technology, which except its reportedly spread-out commercial use, that has already become popular in academia as well. HADOOP provides a powerful programming framework, since the programming concepts are simple and the programmer is freed from all the tedious tasks that one should take care of if he/she would write a distributed piece of code. More details about MAPREDUCE and HADOOP can be found in [19].

Require: Unweighted Graph $G(V, E)$
Require: Sparsification parameter p
Output: $\Delta'(G)$ global triangle estimation
  for each edge e_j **do**
    Toss a biased coin with success probability p
    if $success$ **then**
      $w(e_j) \leftarrow \frac{1}{p}$
    else
      $w(e_j) \leftarrow 0$
    end if
  end for
  $\Delta'(G) \leftarrow$ TRIANGLECOUNTINGALGORITHM(G)
  return $\Delta'(G)$

Algorithm 1: The DOULION counting framework

Require: Unweighted Graph $G(V, E)$
Require: Sparsification parameter p
Output: $\Delta'(G)$ global triangle estimation
  $\Delta'(G) \leftarrow 0$
  for each edge e_j **do**
    Toss a biased coin with success probability p
    if $success$ **then**
      $w(e_j) \leftarrow \frac{1}{p}$
    else
      $w(e_j) \leftarrow 0$
    end if
  end for
  for $v \in V(G)$ **do**
    for all pairs of neighbors (u, w) of v **do**
      if $(u, w) \in E(G)$ **then**
        if $u < v < w$ **then**
          $\Delta'(G) \leftarrow \Delta'(G) + 1$
        end if
      end if
    end for
  end for
  $\Delta'(G) \leftarrow \Delta'(G) * \frac{1}{p^3}$
  return $\Delta'(G)$

Algorithm 2: The DOULION-NODEITERATOR algorithm

3. PROPOSED METHOD

In this section we present the proposed method, analyze it and provide the reader with several interesting -at least in our opinion- observations.

3.1 Algorithm

Our algorithm DOULION is a "friend" rather than a competitor of the other triangle counting algorithms. Furthermore, it is very useful and applicable in all possible scenarios: a) the graph fits in the main memory, b) the size of the graph exceeds slightly the available memory, c) the size of the graph exceeds the available memory significantly. The general framework of the proposed method is shown in algorithm 1. DOULION tosses a coin for each edge. It keeps the edge with probability p and with probability $1-p$ it deletes it. Then each triangle in the resulting graph G' counts as $\frac{1}{p^3}$ triangles. An equivalent way of viewing this procedure is the following:

- Reweight an edge if the edge "survives" with weight equal to $\frac{1}{p}$
- Count each triangle as the product of the weights of the edges comprising the triangle. Since the initial graph G is unweighted each triangle is counted as $(\frac{1}{p})^3 = \frac{1}{p^3}$.

After the tossing-coin stage, *any* triangle counting algorithm can be applied to the obtained graph G'. Algorithm 2 shows the instantiation of the DOULION triangle counting framework using the NODEITERATOR as the triangle counting black box, which was described in section 2. However, in case that even after the sparsification the resulting graph cannot fit into the main memory, a streaming or a semi-streaming algorithm should be preferred instead as the black box.

Observe that since we assume that the input graph G is unweighted all edges in G' will have the same weight. Therefore we can still store efficiently G' just as if it were unweighted plus the parameter p.

3.2 Analysis of DOULION

3.2.1 Mean and Variance

We first show that the expected number of triangles in G' is the number of triangles Δ in the initial graph G. For each triangle in the initial graph, we attach an indicator variable δ_i, $i = 1..\Delta$. Therefore $\delta_i = 1$ if the i-th triangle[1] exists in G', otherwise $\delta_i = 0$. Let X be the random variable that denotes DOULION 's triangles' estimate.

THEOREM 1 (DOULION EXPECTED VALUE). *The expected number of triangles in G' is equal to the actual number of triangles in G: E[X]=Δ*

PROOF. We have that the random variable X is the sum of the indicator variables multiplied by $\frac{1}{p^3}$. By simple properties of the expectation we get the following: E[X]=E[$\sum_{i=1}^{\Delta} \frac{1}{p^3}\delta_i$]= $\sum_{i=1}^{\Delta}$E[$\frac{1}{p^3}\delta_i$]= $\frac{1}{p^3}\sum_{i=1}^{\Delta}$E[$\delta_i$]=$\frac{1}{p^3}\sum_{i=1}^{\Delta} p^3$= Δ □

THEOREM 2 (DOULION VARIANCE). *Let Δ be the total number of triangles in G. The variance is equal to:*

$$Var(X) = \frac{\Delta(p^3-p^6)+2k(p^5-p^6)}{p^6}$$

where k is the number of pairs of triangles that are not edge disjoint.

PROOF. We have that our estimate is a sum of identically distributed but not independently random indicator variables of whether a triangle in the initial graph "survives". The reason that the indicator variables are not independent is shown in figure 3.2.1. The indicator variables δ_i and δ_j for the i-th and j-th triangle are not independent because when the edge that they share does not "survive" then both of them become 0. On the other hand the indicator variables δ_k and δ_p are independent.

Now, by the definition of the variance of a random variable and its basic properties:

$$Var(X) = Var(\frac{1}{p^3}\sum_{i=1}^{\Delta}\delta_i) = \frac{1}{p^6}\sum_{i=1}^{\Delta}\sum_{j=1}^{\Delta} Cov(\delta_i, \delta_j) \quad (1)$$

Now we break up the above summation. There are Δ^2 terms in this sum. Δ of them are the variances of the indicator variables, therefore we get $\Delta(p^3 - p^6)$. The rest $2\binom{\Delta}{2}$ terms correspond to

[1]There is no ordering of triangles. The i-th term refers to the i-th triangle of any random ordering of the triangles in graph G.

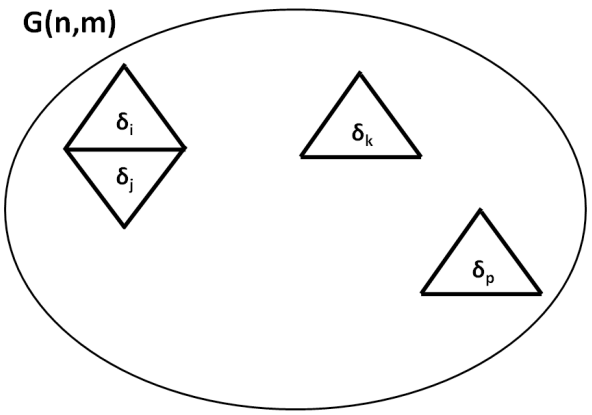


Figure 2: The cases should be considered when estimating the variance of DOULION. These are determined by whether the triangles are edge-disjoint or not.

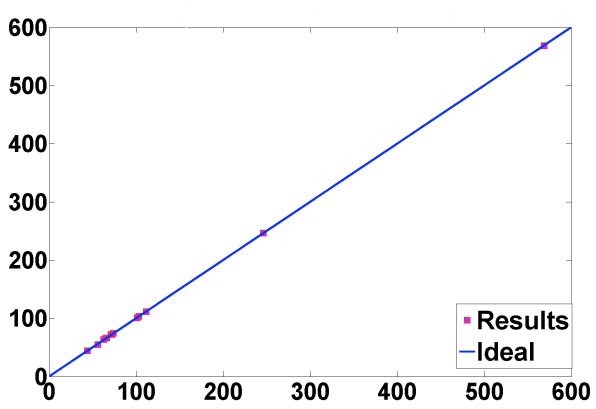


Figure 3: Real Inverse Epidemic Threshold (λ_1) vs. our estimate for 14 different datasets . As we see, the estimates are almost ideal, in most cases differing in the second decimal digit. Similar results hold for other graphs as well.

the pairs of indicator variables. Let k out of $\binom{\Delta}{2}$ pairs of indicator variables correspond to triangles that share one edge. In that case $Cov(\delta_i, \delta_j) = p^5 - p^6$. For the rest $\binom{\Delta}{2} - k$, terms $Cov(\delta_p, \delta_q) = p^6 - p^6 = 0$.

Therefore, we get:

$$Var(X) = \frac{1}{p^6}\left(\Delta(p^3 - p^6) + 2k(p^5 - p^6)\right) \quad (2)$$

□

Using the second moment method ([2]) we get the following theorem.

THEOREM 3. $Pr(|X - \Delta| \geq \epsilon\Delta) \leq \frac{(p^3 - p^6)}{p^6\epsilon^2\Delta} + 2k\frac{(p^5 - p^6)}{p^6\epsilon^2\Delta^2}$

PROOF. By applying Chebyshev's inequality, we get: $Pr(|X - \Delta| \geq \epsilon\Delta) \leq \frac{Var(X)}{\epsilon^2\Delta^2}$ and by substituting the formula for the variance from theorem 2 we get the bound. □

This theorem gives a first insight in the performance of DOULION. The probability that our estimate is away from the real number of triangles by some factor ϵ depends on the number of triangles in the graph as well as the structure of the graph and of course on the sparsification value p in the following way: the larger the number of triangles in the graph, the probability to obtain a good estimate increases. Also, the more edge-disjoint triangles exist in the graph, the better the estimate is. Finally, as $p \to 0$ the quality of the estimate gets worse, as expected.

3.2.2 *Speedup*

Consider now a simple triangle listing algorithm, namely the node iterator which was described in Section 2. If R is its running time after the removal of edges then $R = \sum_{v=1}^{k} D(v)^2$ where $D(v) =$ degree of vertex v after coin-tossing, hence

$$E[R] \sim p^2\left[\sum_v (\deg v)^2\right]. \quad (3)$$

Hence the expected speedup is $\frac{1}{p^2}$.

3.3 Random Sampling

Let's consider the interesting case of a graph that is so large that exceeds the available main memory significantly. A well-known technique to select k random records sequentially from a file that resides in a hard disk is the rejection method [29]. More sampling algorithms can be found in the same work [29] and in [18]. Observe that the number of disk pages fetched may in the worst case be equivalent to performing a sequential scan over the file. However, if k is significantly smaller than the size of the file then we expect to have significant savings with the sampling approach. In our case, where we assume that the graph is represented as a stream of edges or equivalently resides in an edge file, e.g. a file whose each line is of the form (endpoint$_1$, endpoint$_2$), $k \approx mp$.

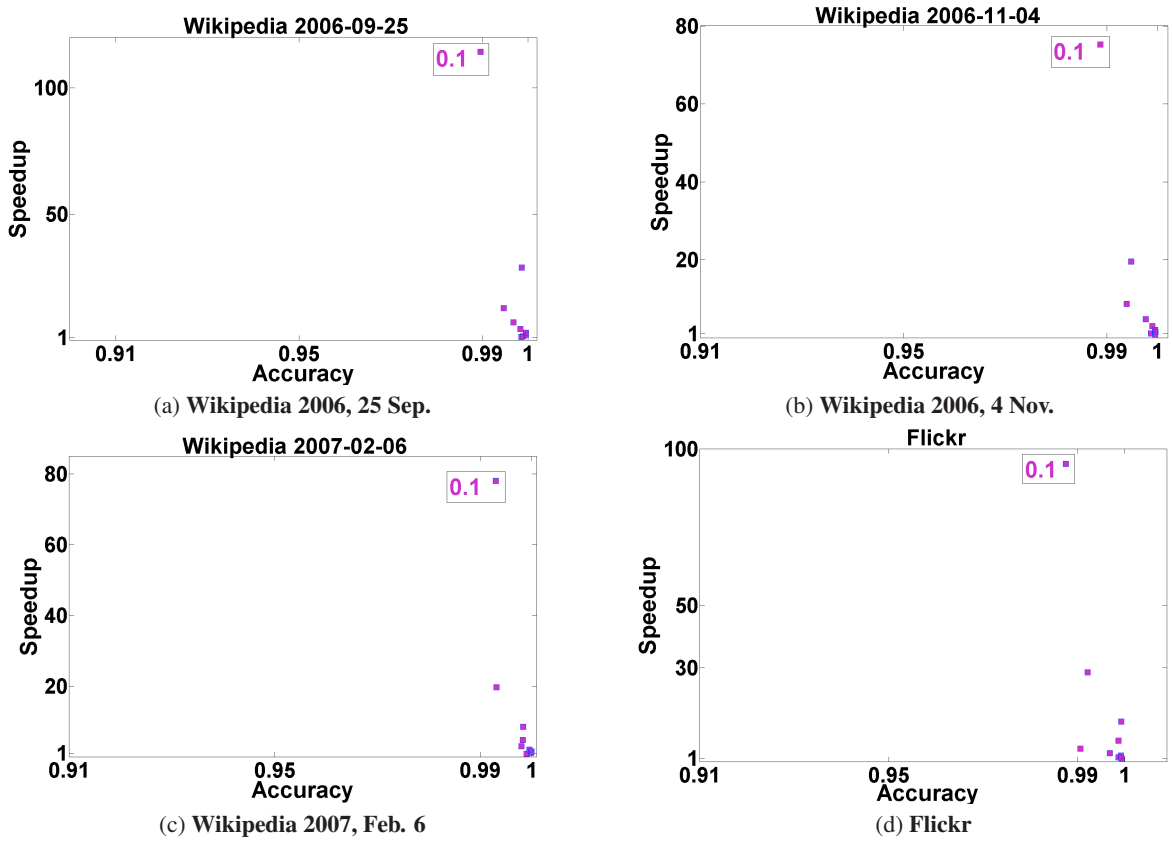
3.4 A Pleasant Side-effect: Preserving the Epidemic Threshold

As shown in [27] the number of triangles is equal to the sum of the cubes of the eigenvalues divided by six. Given the spectra properties observed in many real-world networks one can approximate the number of triangles in the graph just by using few eigenvalues. Achlioptas and McSherry showed in [1] that one can "throw " away many of the elements of a matrix and still keep the top eigenvalues the same. This is an observation used in [28] where a faster version of the algorithm in [27] is presented.

Given the aforementioned observation, the top adjacency eigenvalue of G' will be very close to the top one of G. This is an interesting approach since the top eigenvalue of the adjacency matrix representation of any graph is closely related to the epidemic threshold [30]. Therefore, DOULION has the effect of not only preserving in expectation the number of triangles but also approximately the epidemic threshold.

Just for the sake of illustration, figure 3 plots the real epidemic threshold of graph G vs. the estimate, i.e., the epidemic threshold of graph G' for 14 different datasets (Flickr, Epinions, AS Newman, EAT RS, Lederberg, Patents (main), Patents, Internet, HEP-th (new), Journals, AS Oregon, AS CAIDA (3 timestamps)). As we see from the plot, the results are almost ideal, differing in the first or second decimal digit.

3.5 Can we parallelize DOULION?



(a) **Wikipedia 2006, 25 Sep.** (b) **Wikipedia 2006, 4 Nov.**

(c) **Wikipedia 2007, Feb. 6** (d) **Flickr**

Figure 4: Ideal behavior of DOULIONin graphs with several million of edges. We observe that for all p values ranging from 0.1 to 0.9 the estimate of DOULION is strongly concentrated around its mean value, i.e. the real number of triangles in the graph. The speedups are important, ranging from $\approx$ 80 to $\approx$ 130.

We implemented in HADOOP a prototype for the DOULION-NODEITERATOR. As one can easily observe, the sparsification step is trivially parallel. Each mapper receives a subset of edges of the initial graph and tosses a coin for each edge. If the edge survives, the mapper emits the corresponding edge. The JAVA and HADOOP code of our implementations will be open-sourced. [2]

4. EXPERIMENTS

4.1 Experimental Setup

We implemented DOULION-NODEITERATOR in JAVA and in HADOOP. The HADOOP code ran on Erdős-Rényi graphs and on the real-world networks we ran the JAVA piece of code. The experiments ran on a 4GB RAM, Intel(R) Core(TM)2 Duo CPU at 2.4GHz Windows Vista machine (JAVA code) and in M45 (HADOOP code), one of the fifty most powerful supercomputers in the world (480 hosts, each with 2 quad-core Intel Xeon 1.86 GHz, running RHEL5, with 3Tb aggregate RAM, and over 1.5 PetaByte aggregate disk capacity.) after allocating two commodity machines. The graphs we used in our experiments are described in the table 1. The directed ones were made undirected by removing the arcs of the edges and the self-loops -if any- were removed. Most of the datasets we used are publicly available[3].

[2]http://www.cs.cmu.edu/ ctsourak/projects/triangles.htm.

[3]`http://www.cise.ufl.edu/research/sparse/matrices/`

4.2 Experimental Results

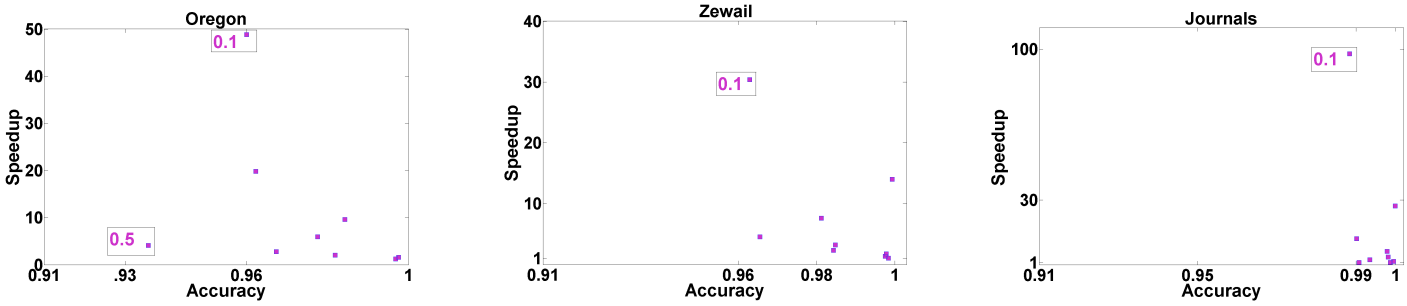
We divide and present the experiments into four different categories: DOULION on large-, medium- and small-sized real world graphs and on Erdős-Rényi. We run DOULION-NODEITERATOR using nine different values for p, ranging from 0.1 to 0.9 with a step of 0.1. All the figures presented in the following refer to a single, random run of DOULION on the graphs.

Large-sized Graphs.

Figures 1 and 4 show the experimental results for the largest real-world graphs we used: the four different snapshots of the Wikipedia Web graph and Flickr. All these networks have size greater than 2M edges. The behavior of DOULION in these graphs is the ideal. The accuracy is always greater than 99% and speedups are significant, ranging from $\approx$ 80 to $\approx$ 130 times faster. As expected, the maximum speedups are obtained for $p = 0.1$. Also observe how more significant the speedups become when moving from $p = 0.2$ to $p = 0.1$. As already mentioned before, observe that the speedup refers to the running time of a straight-forward exact triangle counting method vs. *itself* using DOULION, i.e., NODEITERATOR vs. DOULION-NODEITERATOR. This verifies the fact DOULION is a friend of triangle counting algorithms.

Medium-sized Graphs.

We conducted 158 experiments on medium-sized graphs, whose sized ranged from $\approx$ 40K to $\approx$ 400K edges. Figure 6 shows the



(a) **Oregon** (b) **Zewail** (c) **Journals**

Figure 5: Results of DOULION on the smallest graphs (less than 40K edges) for one random run of DOULION. Again, we observe an excellent performance of DOULION. Compared to the results for the larger graphs, the variance is bigger for the small values of p, though still small. Speedup can be even $\approx$ 100 (Journals).

Nodes	Edges	Description
Real-world Networks		
13,579	37,448	AS Oregon
23,389	47,448	CAIDA AS 2004 to 2008 (means over 151 timestamps)
22,963	48,436	AS NEWMAN
1,634,989	18,540,603	Wikipedia 2005-11-05
2,983,494	35,048,116	Wikipedia 2006-09-25
3,148,440	37,043,458	Wikipedia 2006-11-04
3,566,907	42,375,912	Wikipedia 2007-02-06
27,770	352,285	Hep-th-new
27,240	341,923	Hep-th
8,843	41,532	Lederberg
124	5,972	Journals
13,332	148,038	Reuters
23,219	304,937	Edinburgh Associative Thesaurus (EAT RS)
75,877	405,740	Epinions network
404,733	2,110,078	Flickr
6752	54182	Zewail

Table 1: Summary of real-world networks used.

performance of DOULION on these graphs. For the 150 omitted timestamps/graphs of AS CAIDA, similar results hold as in figure 6(h).

Edinburgh Thesaurus and AS Newman graphs (figures 6(c),(g) exhibit the almost ideal behavior of the large graphs: accuracy always greater than 99% and important speedups. Very close to this behavior, is also behavior of the Epinions (who-trusts-whom), the Reuters' graph and the HEP-TH graph, shown in figures 6(a), (b) and (e). Speedups are still important and accuracy is again high, always more than 97%. In the rest of the graphs (figures 6(d),(f),(h)) results are still satisfactory. However we observe that there is larger variance around the real number of triangles in the graph. Still though, the accuracy is always greater than 96%. The maximum speedup in the case of medium sized graphs can reach 100 times.

Small-sized Graphs.

We used three small graphs to experiment with, AS Oregon, Journals and Zewail. Journals graph exhibits an ideal behavior, just like the large graphs. DOULION gives more than 99% accuracy for all values of p we tried and a speedup of almost 100 times. Oregon and Zewail exhibit larger variance than Journals graph over our single random run. Accuracy is almost always greater than 95% , with the single exception of using $p = 0.5$ in the Oregon graph. However, running DOULION three times, moves these "outlier"-like points closer to 1, just like in all other plots. This was the worst case behavior of DOULION that we saw during our experiments.

Observations .

To sum up, the following observations hold for all the experiments we conducted on real graphs with size ranging from $\approx 6K$ edges (Journals graph) to $\approx 42M$ (Wikipedia 2007):

- Keeping 10% of the edges yields in speedups ranging from $\approx$ 30 to $\approx$ 130 times. Notice that reducing the edges to 10% of the initial amount does not necessarily imply 100x speedup, which is the expected one. On the other hand, as we apply DOULION on large graphs, we are getting closer to the expected speedup.
- Running DOULION three times verifies the fact that the results we obtained were not "random": for most of the graphs the results are almost identical (speedups and accuracies are more or less the same) whereas for few graphs (Oregon and some AS CAIDA timestamps) we see slight larger changes, still though small (e.g. Oregon for p=0.5 gives 93% accuracy). This is expected since the final estimate depends on the initial triangle density: if small, then we expect greater variance in our estimate.

DOULION on Erdős-Rényi $G_{n,\frac{1}{2}}$.

Using our HADOOP implementation we run DOULION on large Erdős-Rényi $G_{n,p}$ graphs. As expected in the case large of random Erdős-Rényi the results are excellent in terms of accuracy for the sparsification values we tested. The reason is the following: after applying DOULION to a $G_{n,p}$ graph with the sparsification parameter equal to 0.1 the result is an Erdős-Rényi $G_{n,p'}$ with $p' = 0.1p$. Therefore, as long as p' is a constant and does not cause any threshold phenomena in the number of cycles in the graph (e.g. $p' = \frac{1}{n}$, see [7]) we have a concentrated estimate around the real number of triangles. The results of running DOULION-NODEITERATOR with $p = 0.1$ on two Erdős-Rényi graphs with 80M and 100M nodes are shown in Table 2. As we see, the speedups are 13.1 and 19.8 respectively for the two graphs and the accuracy in both cases is greater than 99%.

Nodes	Speedup	Accuracy
80M	13.1	99.7
100M	19.8	99.3

Table 2: Results of DOULION on $G_{n,\frac{1}{2}}$ for sparsification value equal to $\frac{1}{2}$.

5. CONCLUSIONS

In this paper we presented DOULION, an algorithm which tosses a coin in order to obtain a smaller, weighted graph in which the number of triangles is very close to the true value. Our contributions can be summarized in the following points:

- DOULION is a "friend" rather than a competitor to other triangle counting algorithms: any other triangle counting triangle algorithm, streaming or not, may use the idea of DOULION as a preprocessing step.
- DOULION is "embarrassingly" parallel, enjoying therefore optimal scale-up in HADOOP.
- We provide a first, basic mathematical analysis which gives some insight in the performance of DOULION with respect to the mean and the variance of the estimator and the expected speedup for the instatiation we used.
- We show that an additional benefit of DOULION is that it maintains the epidemic threshold of the graph.
- We conducted several experiments on real world graphs and for p ranging from 0.1 to 0.9 the accuracy is almost 100% and the speedup can be even ≈130x of a simple exact counting algorithm vs. itself but using DOULION as a first step.

Finally, as a topic of future research, we propose a tighter theoretical analysis that will yield the optimal p, namely the smallest possible one which yields an exponential concentration around the real number of triangles.

6. ACKNOWLEDGEMENTS

The authors would like to thank M.N. Kolountzakis, Tom Bohman and Ioannis Koutis for helpful discussions on the paper. The authors would would like to thank the anonymous reviewers for valuable feedback. This material is based upon work supported by the National Science Foundation under Grants No. CCF-0635257, Also, under the auspices of the U.S. Department of Energy, by the University of California Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344 (LLNL-CONF-404625), subcontracts B579447, B580840. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation, or other funding parties.

7. REFERENCES

[1] D. Achlioptas and F. McSherry. Fast computation of low rank matrix approximation. In *STOC*, 2001.

[2] N. Alon and S. Joel. *The Probabilistic Method*. Wiley Interscience, New York, second edition, 2000.

[3] N. Alon, Y. Matias, and M. 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] N. Alon, R. Yuster, and U. Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997.

[5] Z. Bar-Yosseff, R. 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.

[6] L. Becchetti, P. Boldi, C. Castillo, and A. Gionis. Efficient semi-streaming algorithms for local triangle counting in massive graphs. In *Proceedings of ACM KDD*, Las Vegas, NV, USA, August 2008.

[7] B. Bollobas. *Random Graphs*. Cambridge University Press, 2001.

[8] L. S. Buriol, G. Frahling, S. Leonardi, A. Marchetti-Spaccamela, and C. 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] F. Chung, L. Lu, and V. Vu. Eigenvalues of random power law graphs. *Annals of Combinatorics*, 7(1):21–33, June 2003.

[10] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. In *STOC '87: Proceedings of the nineteenth annual ACM conference on Theory of computing*, pages 1–6, New York, NY, USA, 1987. ACM.

[11] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *OSDI '04*, pages 137–150, December 2004.

[12] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *OSDI*, 2004.

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

[14] I. J. Farkas, I. Derenyi, A.-L. Barabasi, and T. Vicsek. Spectra of "real-world" graphs: Beyond the semi-circle law. *Physical Review E*, 64:1, 2001.

[15] G. Golub and C. Van Loan. *Matrix Computations*. JohnsHopkinsPress, Baltimore, MD, second edition, 1989.

[16] A. Itai and M. Rodeh. Finding a minimum circuit in a graph. In *STOC '77: Proceedings of the ninth annual ACM symposium on Theory of computing*, pages 1–10, New York, NY, USA, 1977. ACM.

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

[18] D. E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming*. Addison-Wesley, Reading, Massachusetts, second edition, 10 Jan. 1981.

[19] R. Lämmel. Google's mapreduce programming model – revisited. *Science of Computer Programming*, 70:1–30, 2008.

[20] M. Latapy. Main-memory triangle computations for very large (sparse (power-law)) graphs. *Theor. Comput. Sci.*, 407(1-3):458–473, 2008.

[21] J. Leskovec and E. Horvitz. Planetary-scale views on an instant-messaging network, Mar 2008.

[22] M. Mcpherson, L. S. Lovin, and J. M. Cook. Birds of a feather: Homophily in social networks. *Annual Review of Sociology*, 27(1):415–444, 2001.

[23] M. Mihail and C. Papadimitriou. the eigenvalue power law, 2002.

[24] M. E. J. Newman. The structure and function of complex networks. *SIAM Review*, 45:167–256, 2003.

[25] T. Sarlos. Improved approximation algorithms for large matrices via random projections. In *FOCS '06: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, pages 143–152, Washington, DC, USA, 2006. IEEE Computer Society.

[26] T. Schank. *Algorithmic Aspects of Triangle-Based Network Analysis*. Phd in computer science, University Karlsruhe, 2007.

[27] C. Tsourakakis. Fast counting of triangles in large real networks, without counting: Algorithms and laws. In *ICDM*, 2008.

[28] C. Tsourakakis, P. Drineas, E. Michelakis, I. Koutis, and C. Faloutsos. Spectral counting of triangles in power-law networks via element-wise sparsification. In *SODA '02: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, 2009.

[29] J. S. Vitter. Faster methods for random sampling. *Commun. ACM*, 27(7):703–718, 1984.

[30] Y. Wang, D. Chakrabarti, C. Faloutsos, C. Wang, and C. Wang. Epidemic spreading in real networks: An eigenvalue viewpoint. In *In SRDS*, pages 25–34, 2003.

[31] S. Wasserman and K. Faust. *Social network analysis*. Cambridge University Press, Cambridge, 1994.

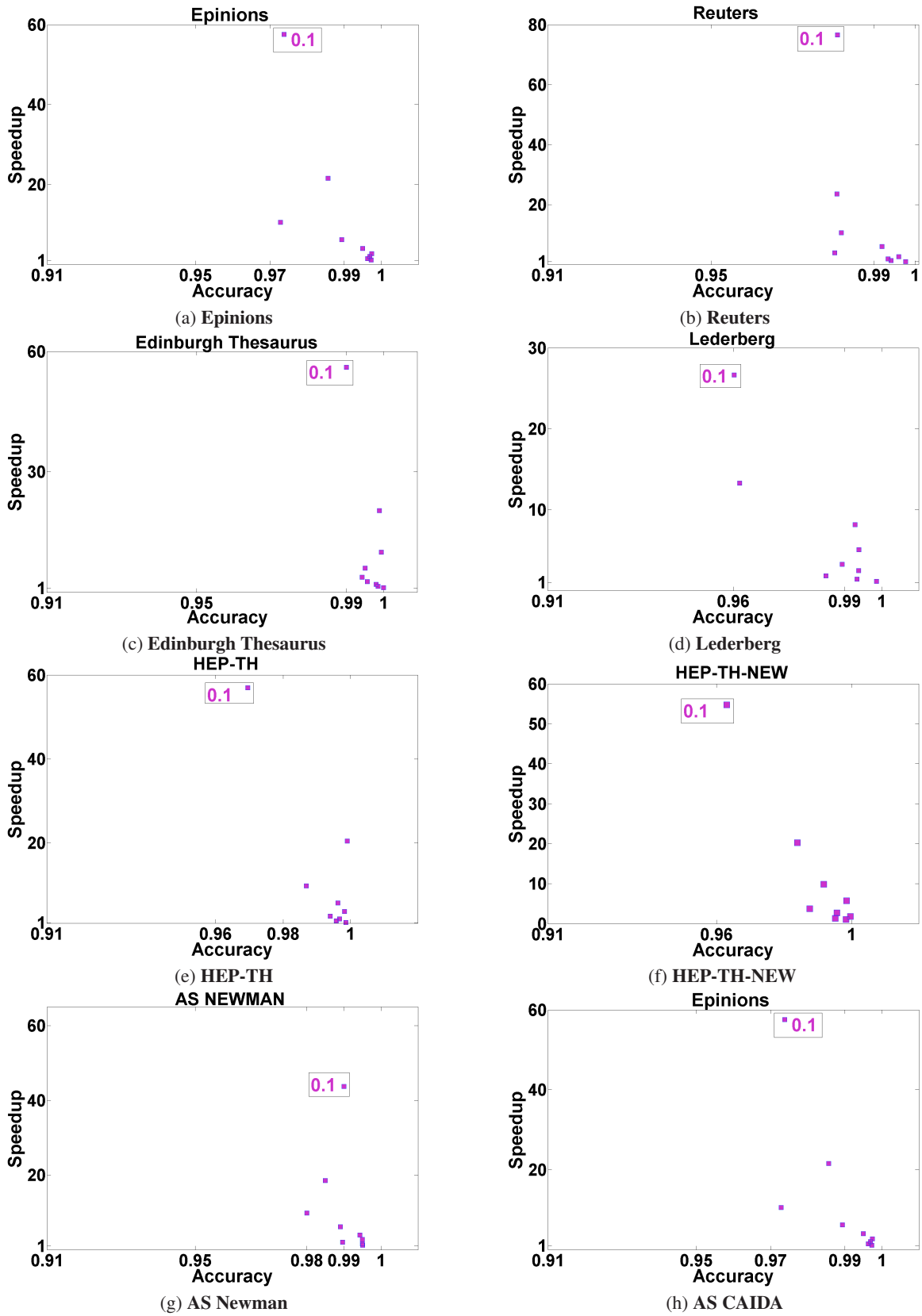


Figure 6: Behavior of DOULION in graphs with several medium sized networks ($\approx$ 40K to $\approx$ 400K edges). As in the case of large and small graphs, we observe that for all p values ranging from 0.1 to 0.9 the estimate of DOULION is strongly concentrated around real number of triangles in the graph. Speedups again are important, ranging from $\approx$ 30 to $\approx$ 60.