

Efficient Semi-streaming Algorithms for Local Triangle Counting in Massive Graphs ^{*}

Luca Becchetti
“Sapienza” Università di Roma
Rome, Italy
becchett@dis.uniroma1.it

Paolo Boldi
Università degli Studi di Milano
Milan, Italy
boldi@dsi.unimi.it

Carlos Castillo
Yahoo! Research
Barcelona, Spain
chato@chato.cl

Aristides Gionis
Yahoo! Research
Barcelona, Spain
gionis@yahoo-inc.com

ABSTRACT

In this paper we study the problem of local triangle counting in large graphs. Namely, given a large graph $G = (V, E)$ we want to estimate as accurately as possible the number of triangles incident to every node $v \in V$ in the graph. The problem of computing the *global* number of triangles in a graph has been considered before, but to our knowledge this is the first paper that addresses the problem of *local* triangle counting with a focus on the efficiency issues arising in massive graphs. The distribution of the local number of triangles and the related local clustering coefficient can be used in many interesting applications. For example, we show that the measures we compute can help to detect the presence of spamming activity in large-scale Web graphs, as well as to provide useful features to assess content quality in social networks.

For computing the local number of triangles we propose two approximation algorithms, which are based on the idea of min-wise independent permutations (Broder et al. 1998). Our algorithms operate in a semi-streaming fashion, using $O(|V|)$ space in main memory and performing $O(\log |V|)$ sequential scans over the edges of the graph. The first algorithm we describe in this paper also uses $O(|E|)$ space in external memory during computation, while the second algorithm uses only main memory. We present the theoretical analysis as well as experimental results in massive graphs demonstrating the practical efficiency of our approach.

^{*}Luca Becchetti was partially supported by EU Integrated Project AEOLUS and by MIUR FIRB project N. RBIN047MH9: “Tecnologia e Scienza per le reti di prossima generazione”. Paolo Boldi was partially supported by the MIUR COFIN Project “Linguaggi formali e automi” and by EU Integrated Project DELIS.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

KDD'08, August 24–27, 2008, Las Vegas, Nevada, USA.
Copyright 2008 ACM 978-1-60558-193-4/08/08 ...\$5.00.

Categories and Subject Descriptors

H.3.3 [Information Systems]: Information Search and Retrieval

General Terms

Algorithms, Measurements

Keywords

Graph Mining, Semi-Streaming, Probabilistic Algorithms

1. INTRODUCTION

Graphs are a ubiquitous data representation that is used to model complex relations in a wide variety of applications, including biochemistry, neurobiology, ecology, social sciences, and information systems. Defining new measures of interest on graph data and designing novel algorithms that compute or approximate such measures on large graphs is an important task for analysing graph structures that reveal their underlying properties.

In this paper we study the problem of counting the local number of triangles in large graphs. In particular, we consider undirected graphs $G = (V, E)$, in which V is the set of nodes and E is the set of edges. For a node u we define $S(u)$ to be the set of neighbors of u , that is, $S(u) = \{v \in V : e_{uv} \in E\}$, and let the degree of u be $d_u = |S(u)|$. We are then interested in computing, for every node u , the number of triangles incident to u , defined as:

$$T(u) = \frac{1}{2} |\{e_{vw} \in E : e_{uv} \in E, e_{uw} \in E\}|.$$

The problem of counting triangles also translates into computing the local clustering coefficient (also known as transitivity coefficient). For a node u , the local clustering coefficient is defined as $\frac{2T(u)}{d_u(d_u-1)}$, that is, the ratio between the number of triangles and the largest possible number of triangles in which the node could participate.

Note that the problem of estimating the overall (global) number of triangles in a graph has been studied already, see e.g. [2, 10]; here we deal with the problem of estimating the (local) number of triangles of all the individual nodes in the graph simultaneously.

We motivate our problem definition by showing how the local triangle computation can be used in a number of interesting applications. Our first application involves spam detection: we show that the distribution of the local clustering coefficient can be an effective feature for automatic Web-spam detection. In particular, we study the distribution of the local clustering coefficient and the number of triangles in large samples of the Web. Results show that these metrics, in particular the former, exhibit statistical differences between normal and spam pages and are thus suitable features for the automatic detection of spam activity in the Web.

Next we apply our techniques to the characterization of content quality in a social network, in our case the Yahoo! Answers community. Following a suggestion from the study of social networks in reference [32], that the type and quality of content provided by the agents is related to the degree of clustering of their local neighborhoods, we perform a statistical analysis of answers provided by users, studying the correlation between the quality of answers and the local clustering of users in the social network.

In addition to the ones we consider, the efficient computation of the local number of triangles and local clustering coefficient can have a larger number of other potential applications, ranging from the analysis of social or biological networks [29] to the uncovering of thematic relationships in the Web [16].

For computing the local number of triangles we propose two approximation algorithms, which rely on well established probabilistic techniques to estimate the size of the intersection of two sets and the related Jaccard coefficient [6, 8, 9]. Our algorithms use an amount of main memory in the order of the number of nodes $O(|V|)$ and make $O(\log |V|)$ sequential scans over the edges in the graph.

Our first algorithm is based on the approach proposed in [7, 8, 9], which uses min-wise independent hash functions to compute a random permutation of an ordered set. In our case, this is the (labeled) set of nodes in the graph. In practice, to increase efficiency, instead of hash functions we simply use a random number generator to assign binary labels to nodes. Doing this can in principle lead to collisions (i.e., we might have subsets of nodes with the same label). We provide a quantitative analysis of this approach, characterizing the quality of the approximation in terms of the Jaccard coefficient and the role of collisions. A similar analysis had been sketched in [6].

We then propose a second algorithm that maintains one counter per node in main memory—as opposed to the first algorithm, which requires one counter for each edge. In practice, our second algorithm allows to perform the computation in main memory, thus achieving a considerable speed up. In particular, the processing time is almost halved, while the accuracy is still comparable or sometimes even better than the first algorithm. This is achieved by using a new, simpler, linear function to approximate the Jaccard coefficient of two sets. As a theoretical contribution, we assess the performance of this second algorithm in the framework used to analyze the first one.

We support our findings and analysis by experimental results. In particular, we use our algorithms to estimate the distributions of the number of triangles and of the clustering coefficient in medium and large samples of the Web graph. To the best of our knowledge, this is the first time efficient

(semi-streaming) approximation algorithms for counting triangles are described.

The rest of the paper is organized as follows. In the next section we review the related work and in Section 3 we introduce the model of computation and the notation that we will be using throughout the paper. Section 4 describes how to approximate the intersection of two sets using pairwise independent permutations, as described in [8]. Section 5 presents our first algorithm, and Section 6 the main-memory-only algorithm. The last section presents our conclusions and outlines future work.

2. RELATED WORK

Computing the clustering and the distribution of triangles are important to quantitatively assess the community structure of social networks [29] or the thematic structure of large, hyperlinked document collections, such as the Web [16].

There has been work on the exact computation of the number of triangles incident to each node in a graph [1, 3, 25]. The brute-force algorithm for computing the number of triangles simply enumerates all $\binom{|V|}{3}$ triples of nodes, and thus it requires $O(|V|^3)$ time. A more efficient solution for the local triangle counting problem is to reduce the problem to matrix multiplication, yielding an algorithm with running time $O(|V|^\omega)$, where currently $\omega \leq 2.376$ [14]. If in addition to counting one wants to list all triangles incident to each node in the graph, variants of the “node iterator” and “edge-iterator” algorithms can be used. A description and an experimental evaluation of those “iterator” algorithms can be found in [30]; however, their running time is $O(|V|d_{\max}^2)$ and $O(\sum_{v \in V} d_v^2)$, respectively. For the datasets we consider—very large number of nodes and high-degree nodes due to skewed degree distributions—such exact algorithms are not scalable, thus in this paper we resort to approximation algorithms.

In [13] the authors propose a streaming algorithm that estimates the global number of triangles with high accuracy, using an amount of memory that decreases as the number of triangles increases. This result has been improved in [10]. We remark that, differently from [13, 10], in this paper we are interested in estimating the local clustering coefficient (and the number of triangles) for all vertices at the same time.

Min-wise independent permutations have been proposed by Broder et al. as a way to estimate the size of the intersection of two sets and the related Jaccard coefficient. Together with the technique of shingles they provide a powerful tool to detect near duplicates in large document collections and the Web in particular [9, 6, 7]. Implementing min-wise independent permutations is infeasible in practice, since they require exponential space [8]. In recent years, families of linear hash functions have been proposed that implement min-wise independent permutations approximately [24, 4]. As explained further in this paper, in order to save computational time we do not use hash functions directly, but rather a pseudo-random generator. This can bring to collisions, but we show that their impact is negligible in practice.

The probabilistic estimation techniques we use have been considered in the past to solve related problems. In [20], the authors use the techniques of shingles and linear hashing to

discover subsets of Web pages that share significant subsets of their outlinks, thus extending and making the discovery of cyber-communities in the Web computationally more efficient, in the spirit of [27]. Finally, in [19], the authors apply similar techniques to produce indices of page similarity that extend SimRank [26].

3. PRELIMINARIES

3.1 Semi-streaming graph algorithms

Given the very large size of the data sets used in Web Information Retrieval, efficiency considerations are very important. For concreteness, the total number of nodes $N = |V|$ in the Web that is indexable by search engines is in the order of 10^{10} [21], and the typical number of links per Web page is between 20 and 30.

This fact imposes severe restrictions on the computational complexity of feasible algorithmic solutions. A first approach to modeling these restrictions might be the *streaming model* of computation [23], which however imposes limitations that are too severe for the problem at hand. Instead, we focus on building algorithmic solutions whose space and time requirements are compatible with the *semi-streaming model* of computation [17, 15]. This implies a semi-external memory constraint [31] and thus reflects many significant limitations arising in practice. In this model, the graph is stored on disk as an adjacency list and no random access is possible, i.e., we only allow sequential access. Every computation involves a limited number of sequential scans of the data stored in secondary memory [22].

Our algorithms also use an amount of main memory in the order of the number of nodes, whereas an amount of memory in the order of the number of edges may not be feasible. We assume that we have $O(N \log N)$ bits of main (random access) memory, i.e., in general there is enough memory to store some limited amount of data about each vertex, but not to store the links of the graph in main memory. We impose as a further constraint that the algorithm should perform at most $O(\log N)$ passes over the data stored on secondary storage.

For comparison, suppose we want to measure the number of triangles in a graph in a naïve way. This would imply loading the lists of neighbors of each node in the graph in main memory to be able to count the number of triangles directly. This would need $O(|E| \log |V|)$ bits of main memory which is impractical in general. As to this point, note that in many data sets arising in practice, in particular some of the ones we consider in the experiments, we have $|E| = \Omega(|V| \log |V|)$.

3.2 Counting triangles

Considered an undirected graph (possibly a symmetrized version of a Web graph) and a vertex u , denote by $S(u)$ the set of u 's immediate neighbors. Now notice that, for every edge $uv \in E$, the number of triangles to which both u and v belong is $|S(u) \cap S(v)|$. So, the overall number of triangles $u \in V$ is participating in is $\sum_{v \in S(u)} |S(u) \cap S(v)|$. As a result, the basic building block of our approach is an algorithm to estimate the size of the intersection of two sets.

In the next section, we revisit the general technique [8, 9, 6, 7] to estimate the Jaccard coefficient and thus the size of the intersection of two sets A and B , defined over the same

Table 1: Datasets used in the experiments.

Collection	Domain	Year	Nodes	Edges
WEBBASE-2001	various	2001	118M	1737M
IT-2004	.it	2004	41M	2069M
EU-2005	.eu.int	2005	862K	33M
UK-2006-05	.uk	2006	77M	5294M
Answers	social net	2007	6M	277M

universe which we assume, without loss of generality, to be $[n] = \{0, \dots, n-1\}$ and where $n = 2^k$ for some suitable k .

3.3 Datasets

We ran most of our experiments on three medium-sized crawls gathered by the Laboratory of Web Algorithmics, University of Milan (<http://law.dsi.unimi.it/>); the graphs were symmetrized and loops were not considered in the computations. We used the WebGraph framework [5] to manipulate the graphs in compressed form. The particular collections we used are listed in Table 1. Note that, at least for some of the collections we consider, $|E|$ is expected to grow as $\Omega(|V| \log |V|)$. Furthermore, consistently with the empirical observations in [28], the average number of edges per node increases over the years. The dataset UK-2006-05 is the crawl that was labeled by a team of volunteers for creating a Web-spam collection [11] so we have labels of non-spam/spam for a large set of hosts in that collection. The distribution of the number of triangles in the smaller graph EU-2005 is shown in Figure 1 and follows a power law.

In addition to the graphs from web crawls, we also used a subgraph from Yahoo! Answers (<http://answers.yahoo.com/>), a question-answering portal. In the graph, each node represents a user, and a link between two users indicates that one of the users has answered a question asked by the other user. In the system, users can choose among the answers received which one is the best answer, and in the graph, we have identified the users who provide a high proportion of “best answers” to the questions they answer.

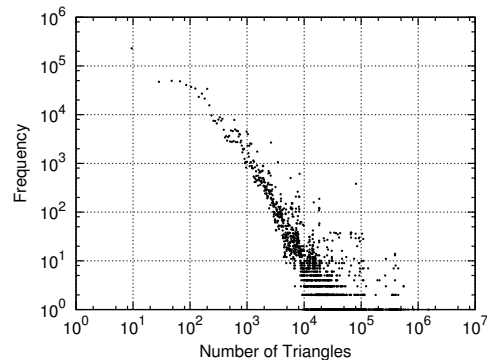


Figure 1: Distribution of the number of triangles in the EU-2005 graph.

4. ESTIMATING SET INTERSECTION

Without loss of generality, we consider subsets of the universe $[n] = \{0, \dots, n-1\}$. We measure the overlap of two sets using the Jaccard coefficient: $J(A, B) = \frac{|A \cap B|}{|A \cup B|}$.

A very simple and elegant technique to estimate the Jaccard coefficient has been proposed in several equivalent forms by Broder et al. [6, 7, 8, 9]. Assume we are able to choose a permutation $\pi(\cdot)$ mapping $[n]$ onto itself uniformly at random. For every $X \subseteq [n]$, denote by $\pi(X)$ the set of the images of elements in X when $\pi(\cdot)$ is applied and let $\min(\pi(X))$ denote their minimum. Then it can be shown [7] that (i) for every $a \in A \subseteq [n]$, $\Pr[a = \arg \min(\pi(A))] = 1/|A|$; (ii) for every $A, B \subseteq [n]$: $\Pr[\min(\pi(A)) = \min(\pi(B))] = J(A, B)$. This property immediately yields a technique to estimate $J(A, B)$. The algorithm consists in performing m passes over the data. At each pass, one permutation $\pi(\cdot)$ among the $n!$ possible ones is picked uniformly at random and then $\min(A)$ is computed and compared with $\min(B)$. Whenever they match, a counter is updated. Let C_m be the counter's value after m passes. Our estimation of $J(A, B)$ is C_m/m .

Unfortunately, generating permutations uniformly at random requires exponential space [8]. In practice, suitable families of linear hash functions are used (e.g. see [24, 4]).

In this paper, in order to increase the speed of computation, we use a slight modification of this approach, simply assigning random labels to the graph's vertices. As long as labels are sufficiently random and collisions not too frequent, we are able to approximate the Jaccard coefficient satisfactorily. In practice, we used the Mersenne Twister, a pseudo-random number generator, which is a fast generation algorithm for obtaining high-quality pseudo-random numbers.

Figure 2 describes the algorithm's pseudo-code, which is exactly the standard one given for example in [7], except for the use of random labels. As to the notation used in the pseudo-code, $\mathbf{l}(j)$ is a k -bit integer label for every item $j \in [n]$ while, for $A \subseteq [n]$, $\mathbf{L}(A) = \min_{j \in A} \mathbf{l}(j)$.

Require: sets $A, B \subseteq [n]$, integer m , k bits

- 1: **for** $i : 1 \dots m$ **do**
- 2: For every $j \in [n]$, set $\mathbf{l}(j)$ to a value drawn uniformly at random between 0 and $2^k - 1$
- 3: **COMPUTE** $\mathbf{L}(A)$ **AND** $\mathbf{L}(B)$
- 4: **if** $(\mathbf{L}(A) == \mathbf{L}(B))$ **then**
- 5: **count** \leftarrow **count** + 1
- 6: **return** **estimate** \leftarrow $(\text{count}/(\text{count} + m))(|A|+|B|)$

Figure 2: Basic algorithm for the estimation of the intersection of two sets.

Define the following variables: $W_i = 1$ if and only if, in the i -th iteration, $\mathbf{L}(A) = \mathbf{L}(B)$ and $W = \sum_{i=1}^m W_i$. Set $X = |A \cap B|$. Our estimator of X is $\bar{X} = W/(W + m)(|A| + |B|)$. In fact, the labeling step might assign the same label to multiple vertices. This means that, in each iteration of the algorithm above, the probability that $\mathbf{L}(A) = \mathbf{L}(B)$ is not exactly equal to $J(A, B)$, as would be the case if we used min-wise independent permutations [8]. For the sake of completeness, we show that, as long as labels are reasonably random, the trivial labeling scheme we use allows us to estimate $J(A, B)$ with good accuracy, collisions having a negligible impact. This is stated in the next result, whose proof follows the lines of those given in [9, 6, 7] and will be given in the full version of the paper. We present this result here for the sake of completeness, since it considers the role of collisions (an aspect only sketched in [6]).

THEOREM 1. *For every $\epsilon > 0$ and for every number m of iterations:*

$$\Pr[|\bar{X} - X| > \epsilon X] \leq 2e^{-\frac{2}{3}mJ(A,B)} + \frac{m|A \cup B|}{2^k - 1}.$$

In practice, this result states that our estimation of $|A \cap B|$ differs from the true value by more than a constant factor with a probability that exponentially decays with m and $J(A, B)$, while the worst case impact of collisions is summarized in the second term, which is $o(1)$ as long as $k = \Omega(\log n + \log m)$, m typically being in the order of a few tenths.

In the next section, we describe how to apply the same techniques for estimating the number of triangles.

5. ESTIMATING TRIANGLE COUNT

In this section we describe an approximating algorithm for counting the number of triangles for each node in the graph. The idea is to compute an approximation $\overline{T}(u)$ of the number of triangles $T(u)$ for all vertices in the graph.

5.1 Algorithm

The algorithm for computing the number of triangles is written in pseudo-code in Figure 3 and explained in the next paragraphs. The notation used in the pseudo-code is as follows: $G = (V, E)$ is an undirected graph, $S(u)$ is the set of neighbors of vertex u , $h_p(u)$ denotes the random k -bit label for node u .

Require: graph $G = (V, E)$, number of iterations m , number of bits k

- 1: $Z \leftarrow 0$
- 2: **for** $p : 1 \dots m$ **do** {This reads the graph $2m$ times}
- 3: **for** $u : 1 \dots |V|$ **do** {Initialize node labels and min}
- 4: $h_p(u) \leftarrow k$ random bits
- 5: $\min(u) \leftarrow +\infty$
- 6: **for** $src : 1 \dots |V|$ **do** {Compute minima}
- 7: **for all** links from src to $dest$ **do**
- 8: $\min(src) \leftarrow \min(\min(src), h_p(dest))$
- 9: **for** $src : 1 \dots |V|$ **do** {Compare minima}
- 10: **for all** links from src to $dest$ **do**
- 11: **if** $\min(src) == \min(dest)$ **then**
- 12: $Z_{src,dest} \leftarrow Z_{src,dest} + 1$
- 13: **for** $src : 1 \dots |V|$ **do** {Compute number of triangles}
- 14: $\overline{T}(src) \leftarrow 0$
- 15: **for all** links from src to $dest$ **do**
- 16: $\overline{T}(src) \leftarrow \overline{T}(src) + \frac{Z_{src,dest}}{Z_{src,dest} + m} (|S(src)| + |S(dest)|)$
- 17: $\overline{T}(src) \leftarrow \overline{T}(src)/2$
- 18: **return** $\overline{T}(\cdot)$

Figure 3: Algorithm for estimating the number of triangles of each node. The counters $Z_{\cdot, \cdot}$ are kept on external memory and updated sequentially.

The algorithm performs m passes. At the beginning of each pass p , a new random vector $h_p(\cdot)$ is created. Each pass consists of two reads of the graph. In the first read of the graph, at each node we store the minimum label among those of the neighbors of that node. In the second read of

the graph, we check, for each edge, if the two minima at the endpoints of the edge are equal; in such a case, one counter Z_{\cdot} , for each edge is increased.

After the m passes, an estimation of the number of triangles of each node is computed as:

$$\overline{T(u)} = \frac{1}{2} \sum_{v \in S_u} \frac{Z_{uv}}{Z_{uv} + m} (|S(u)| + |S(v)|).$$

The algorithm is feasible because the counters Z_{uv} , which make most of the memory usage, are accessed sequentially and can be kept on secondary memory. The time complexity of the algorithm is $O(m|E|)$. The main memory usage is $O(k|V|)$ bits, basically for storing the node labels and the minima; a natural choice for k is $\log(|V|)$. The secondary memory usage is $O(|E| \log m)$ bits of temporary space which is less than the space required to store the graph in uncompressed form. The space required in secondary memory is read and written sequentially once for each pass.

The quality of the approximation only depends on local properties of the graph, and does not vary as the graph grows in size. In particular, every term in the sum above has an accuracy that is described by Theorem 1, where $A = S(u)$ and $B = S(v)$. So, as stated in the previous section, the approximation improves with the number of passes, and it depends on the Jaccard coefficient so that for nodes with higher Jaccard coefficient the error is smaller.

Remark. The value of m depends on the desired per-node accuracy. As Theorem 1 shows, a value of m in the order of a few tenths suffices to satisfactorily estimate the size of the intersection of any two neighbourhoods that overlap significantly.

5.2 Experimental results

We first computed the exact number of triangles for a large sample of nodes in main memory. To do this, we proceeded by blocks, keeping in main memory the neighbors of a set of vertices, counting triangles, and then moving to the next block of nodes. We did this for a sample of 4M nodes in each graph (except in the small one EU-2005 in which we were able to sample all the 800K nodes).

We use two similarity measures: Pearson’s correlation coefficient (r) and Spearman’s rank correlation coefficient (ρ) between the approximation and the real value. We also measured the average relative error:

$$\frac{1}{|V|} \sum_u \frac{|T(u) - \overline{T(u)}|}{T(u)}.$$

As a baseline approximation, we assume a constant clustering coefficient C in the graph, known in advance, and estimate the number of triangles of a node u as $C \frac{|S(u)|(|S(u)-1|)}{2}$. For two of the metrics we use for measuring the quality of the approximation below, the value of C is not relevant: Pearson’s correlation coefficient assumes a linear relationship and Spearman’s rank correlation coefficient is not affected by multiplicative factors.

Next we computed the distribution using our algorithm. For a fixed number of bits k , the accuracy of the approximation increases with the number of passes. In Figure 4 we show the error of these approximations in one of the Web graphs; the result for the other Web graphs in Table 1 are equivalent.

Already at 20 passes, involving only 40 sequential reads of the graph, the approximation has $r \geq 0.90$ and $\rho \geq 0.90$.

Looking at Spearman’s rank correlation, which is ≥ 0.85 with 50 iterations for our algorithm, we can see that the baseline algorithm provides a better approximation of the ordering of the nodes by number of triangles in IT-2004, EU-2005 and UK-2006-05. This fact indicates that the overall ordering is dominated by the degree of the nodes involved. However, the correlation coefficient of the baseline approximation is very low (below 0.5, and below 0.1 in UK and WebBase) while the correlation coefficient of the proposed algorithms is above 0.9.

Remark. For the sake of brevity, we only mention here that our algorithms show that the distribution of the number of triangles follows a power law, as shown in Figure 1. The same observation was also made in [16] for Web samples of smaller size.

6. ESTIMATING TRIANGLE COUNT IN MAIN MEMORY

This section describes a modification of previous algorithm that does not make use of external memory for the computation.

Observe that, in the final step of the algorithm presented in Section 5, we computed an estimation of the number of triangles of a node as:

$$\overline{T(u)} = \frac{1}{2} \sum_{v \in S_u} \frac{Z_{uv}}{Z_{uv} + m} (|S(u)| + |S(v)|)$$

in which Z_{uv} is the number of minima that were the same between u and v during the m passes, so $0 \leq Z_{uv} \leq m$.

To avoid the use of external memory, instead of keeping one counter for each edge, we can use one counter for each node, by approximating the number of triangles incident to a vertex u as:

$$\overline{\overline{T(u)}} = \frac{1}{2} \sum_{v \in S_u} \frac{Z_{uv}}{\frac{3}{2}m} (|S(u)| + |S(v)|).$$

The algorithm that uses this approximation is given in Figure 5 and it is explained in the next paragraphs. The proof that it estimates the triangle count with good accuracy is given in the next subsection.

This algorithm is similar in spirit to the one shown in Figure 3, but removing Z_{uv} from the denominator in the expression of $\overline{\overline{T(u)}}$ allows to maintain one counter per node instead of one counter per edge. The algorithm does m passes, each pass consisting of two reads of the graph. In the first read of the graph, at each node we store the minimum hash value of the neighbors of that node. In the second read of the graph, we check, for each edge, if the two minima at the endpoints of the edge ($src, dest$) are equal, and if so a *per-node counter* Z_{src} is increased by $|S(src)| + |S(dest)|$.

After the m passes, an estimation of the number of triangles of each node is computed as:

$$\overline{\overline{\overline{T(u)}}} = \frac{1}{2} \frac{Z_u}{\frac{3}{2}m} = \frac{1}{3m} Z_u.$$

The time complexity of the algorithm is $O(m|E|)$. The main memory usage is $O(k|V|)$ bits, basically for storing the hash functions, minima, and the per-node counters. Secondary memory is accessed only to read the graph.

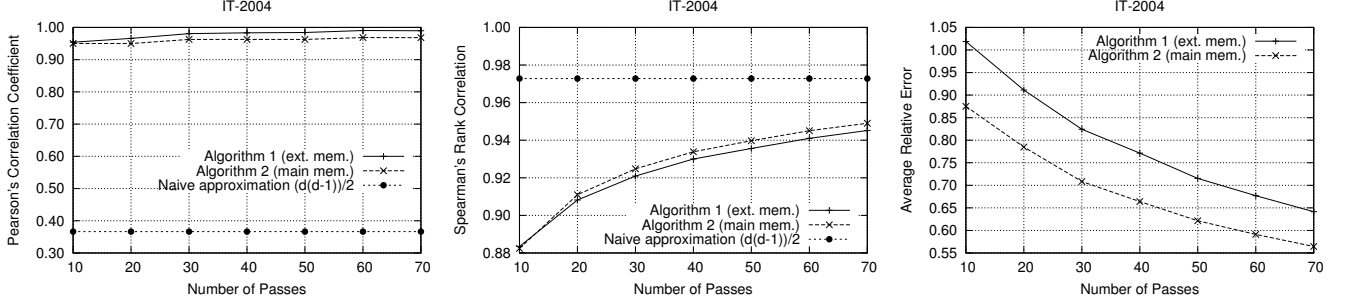


Figure 4: Accuracy of the approximation of the number of triangles using the two algorithms described in the paper (external memory and main memory). Left: Pearson's correlation coefficient. Center: Spearman's rank correlation coefficient. Right: average relative error.

Require: graph $G = (V, E)$, number of iterations m , number of bits k

```

1:  $Z \leftarrow 0$ 
2: for  $p : 1 \dots m$  do {This reads the graph  $2m$  times}
3:   for  $u : 1 \dots |V|$  do {Initialize node labels and min}
4:      $h_p(u) \leftarrow k$  random bits
5:      $\min(u) \leftarrow +\infty$ 
6:   for  $src : 1 \dots |V|$  do {Compute minima}
7:     for all links from  $src$  to  $dest$  do
8:        $\min(src) \leftarrow \min(\min(src), h_p(dest))$ 
9:   for  $src : 1 \dots |V|$  do {Compare minima}
10:    for all links from  $src$  to  $dest$  do
11:      if  $\min(src) == \min(dest)$  then
12:         $Z_{src} \leftarrow Z_{src} + |S(src)| + |S(dest)|$ 
13: for  $u : 1 \dots |V|$  do {Compute number of triangles}
14:    $\overline{T}(src) \leftarrow \frac{1}{3m} Z_u$ 
15: return  $\overline{T}(\cdot)$ 

```

Figure 5: Algorithm for estimating the number of triangles of each node in main memory.

6.1 Analysis

We can give a result similar to that of Theorem 1. Namely, for $u, v \in V$, set $X = |S(u) \cap S(v)|$ and define W as in Section 4. In particular, $W = \sum_{i=1}^m W_i$, with $W_i = 1$ if, during the i -th iteration of the algorithm, the **if** at line 14 of the algorithm of Figure 5 is true for nodes u and v . Finally, define

$$\overline{X} = \frac{W}{1.5m} (|S(u)| + |S(v)|).$$

We have

THEOREM 2.

$$\begin{aligned} & \Pr \left[\left(\overline{X} > \frac{4}{3}(1 + \epsilon)X \right) \cup \left(\overline{X} < \frac{2}{3}(1 - \epsilon)X \right) \right] \leq \\ & \leq 2e^{-\frac{2}{3}mJ(S(u), S(v))} + \frac{m|S(u) \cup S(v)|}{2^k - 1}. \end{aligned}$$

PROOF. For $i = 1, \dots, m$, let $E_i = 1$ if at the i -th iteration there is more than one element achieving the minimum, 0 otherwise and let $E = \sum_{i=1}^m E_i$. Also, set $\overline{W}(i) =$

$(W_i | E = 0)$ and $\overline{W} = \sum_{i=1}^m \overline{W}(i)$. We have:

$$X = |S(u) \cap S(v)| = \frac{\mathbf{E}[\overline{W}]}{\mathbf{E}[\overline{W}] + m} (|S(u)| + |S(v)|).$$

Set $\hat{X} = (\overline{X} | E = 0)$. Since $0 \leq \mathbf{E}[\overline{W}] \leq m$, it is easy to see that we have:

$$\frac{2}{3}X \leq \mathbf{E}[\hat{X}] \leq \frac{4}{3}X.$$

We have:

$$\begin{aligned} & \Pr \left[\left(\overline{X} > \frac{4}{3}(1 + \epsilon)X \right) \cup \left(\overline{X} < \frac{2}{3}(1 - \epsilon)X \right) \right] \\ & < \Pr \left[\left(\left(\overline{X} > \frac{4}{3}(1 + \epsilon)X \right) \cup \left(\overline{X} < \frac{2}{3}(1 - \epsilon)X \right) \right) | E = 0 \right] \\ & + \Pr[E > 0] \\ & < \Pr \left[\left(\hat{X} > \frac{4}{3}(1 + \epsilon)X \right) \right] + \Pr \left[\left(\hat{X} < \frac{2}{3}(1 - \epsilon)X \right) \right] \\ & + \frac{m|S(u) \cup S(v)|}{2^k - 1}, \end{aligned}$$

where the last inequality follows from the definition of \hat{X} . Now:

$$\begin{aligned} & \Pr \left[\left(\hat{X} > \frac{4}{3}(1 + \epsilon)X \right) \right] + \Pr \left[\left(\hat{X} < \frac{2}{3}(1 - \epsilon)X \right) \right] \\ & \leq \Pr \left[|\hat{X} - \mathbf{E}[\hat{X}]| > \epsilon \mathbf{E}[\hat{X}] \right], \end{aligned}$$

where the inequality follows from the above given bounds on $\mathbf{E}[\hat{X}]$ in terms of X . Recalling that, by definition, $\hat{X} = \overline{W}(|S(u)| + |S(v)|)/(1.5m)$ we immediately have:

$$\Pr \left[|\hat{X} - \mathbf{E}[\hat{X}]| > \epsilon \mathbf{E}[\hat{X}] \right] = \Pr \left[|\overline{W} - \mathbf{E}[\overline{W}]| > \epsilon \mathbf{E}[\overline{W}] \right].$$

The rest of the proof now proceeds exactly as in Theorem 1. \square

Remark. As to the choice of m , considerations analogous to those at the end of Section 5 hold.

6.2 Experimental results

In practice, we observe that the second algorithm saves 40% to 60% of the running time. We ran the experiments for the large graphs in a quad-processor Intel Xeon 3GHz with 16GB of RAM. The wall-clock times required for $m = 50$ iterations we observed were:

Graph	Nodes	Edges	Algorithm 1 (ext. mem.)	Algorithm 2 (main mem.)
WB-2001	118M	1.7G	10 hr 20 min	3 hr 40 min
IT-2004	41M	2.1G	8 hr 20 min	5 hr 30 min
UK-2006	77M	5.3G	20 hr 30 min	13 hr 10 min

The experimental results obtained show that, surprisingly, in many cases the accuracy of the main-memory algorithm is even better than the algorithm that uses secondary memory. Figure 4 depicts the results for the case of IT-2004 (experiments on the other datasets have been omitted for lack of space, but have essentially the same behavior).

In the implementation, the number of bits necessary to store each counter depends on the number of iterations and on the link density of the graph. For instance, for WB-2001 we used a Java `int` (32-bits including the sign), but for IT-2004 and UK-2006, a `long` (64 bits including sign) was necessary to avoid overflow. We started to observe overflow after 60 passes in IT-2004 and after 20 passes in UK-2006. We point out that this is independent from the number of nodes in the graph.

7. APPLICATIONS

An efficient algorithm for local triangle counting is not only interesting as an algorithmic contribution. This section describes two applications of the algorithm for helping in information retrieval tasks in large graphs.

7.1 Detecting Web spam

Spam and non-spam pages exhibit different statistical properties, and this difference can be exploited for Web Spam Detection [18]. In this section we test if the number of triangles is a relevant feature for this task.

We used the `WEBSpAM-UK2006` spam collection [11], a public Web Spam dataset annotated at the level of hosts. First we computed the number of triangles for each host in this dataset and plotted the distribution for the non-spam and spam hosts. This is shown in Figure 6. A two-tailed Kolmogorov-Smirnov test indicates that both the number of triangles and the clustering coefficient have distributions that are substantially different in both classes: the larger differences in the cumulative distribution function plot are $D = 0.32$ and $D = 0.34$ respectively.

We also compared the number of triangles and clustering coefficient with a known set of link-based and content-based features for the hosts in this collection [12]. We sorted all the features by computing the χ -squared statistics of each of them with respect to the class label. Using this ranking, the approximated number of triangles was ranked as feature number 60 out of 221, and the approximated clustering coefficient as feature number 14 out of 221; such remarkably high positions make both features well worth being tested as part of a spam detection system.

To complement these results, we estimated the number of triangles *at a page level*, and considered the average and maximum number of triangles in every host; in all cases we had to use the memory-based approximation algorithm to obtain the estimation, since an exact counting was in this case out of question. The results are shown in Figure 7. Also in this case, a two-tailed Kolmogorov-Smirnov proved that the spam and non-spam distributions actually differ from each other: for example, the test in the case of average gave $D = 0.09$ with a p-value of $1.54 \cdot 10^{-7}$.

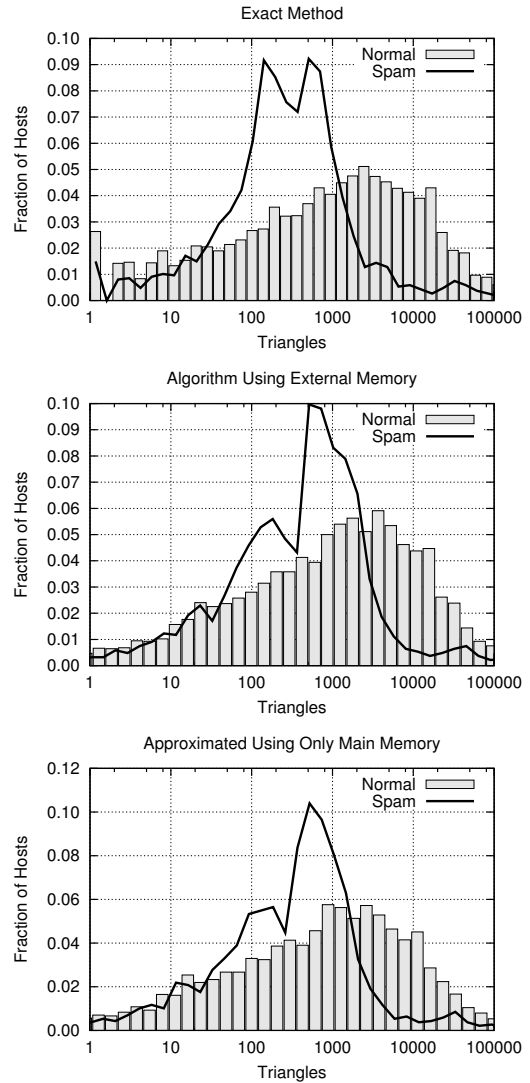


Figure 6: Separation of non-spam and spam hosts in the histogram of triangles, measured using the exact algorithm (top), the approximated algorithm with 50 passes (middle) and the approximated algorithm in main memory with 50 passes (bottom).

7.2 Content quality in social networks

In [32] it is shown that the amount of triangles in the self-centered social network of a user is a good indicator of the role of that user in the community.

Here we perform an exploration trying to verify whether the quality of content provided by a user in a social network is correlated with the local structure of the user in the network. For our dataset, we use a social network extracted from the Yahoo! Answers site. Yahoo! Answers is a community-driven knowledge sharing system that allows users to (i) ask questions on any subject and (ii) answer questions of other users. One notable characteristic of the system is that one answer for each question is selected as the *best answer*, and one of the user attributes is the fraction of the best answers given by that user.

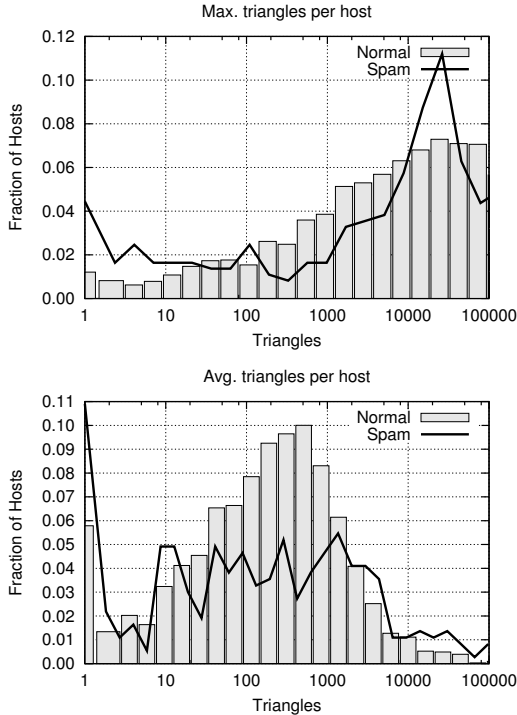


Figure 7: Separation of non-spam and spam hosts in the histogram of triangles computed at page level and maximized/averaged on each host.

We consider an undirected graph $G = (V, E)$, where V is a set of users in the system, and an edge $(u, v) \in E$ denotes that the user u has answered a question posted by user v , or vice versa. For this graph we apply our counting algorithms and we obtain an estimate of the number of triangles at each node, as well as the local clustering coefficient. We focus on a small subset of randomly chosen questions and answers which have been labeled by human judges as “high quality” or “normal”. These questions/answers have originated from a subset of about 9,500 users. Let $H \subseteq V$ be the subset of users who have provided a question or answer of high quality in our sample, corresponding to roughly 30% of the users in this case, and let $N = V \setminus H$ be the rest.

As a proof of concept, we first check if the fraction of best answers for the users differs between the sets H and N . The two distributions are shown in Figure 8, in which one sees that users in the high quality set tend to have higher fractions of best answers. The two-tailed Kolmogorov-Smirnov difference of the two distributions is 0.26, and the null hypothesis is rejected with corresponding p -value equal to $1.1 \cdot 10^{-123}$.

Next we explore the correlation of local structure in the user graph with respect to the labeling of users in the classes H and N . In particular, we examine if the distribution of the number of triangles and the distribution of the local clustering coefficient differ between the sets H and N . The distributions in the case of the numbers of triangles are different. The Kolmogorov-Smirnov test rejects the null hypothesis with difference value equal to 0.12 and p -value equal to $2.9 \cdot 10^{-29}$.

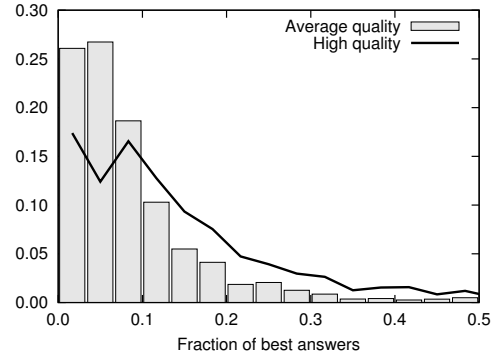


Figure 8: Separation of users who have provided questions/answers of high quality with users who have provided questions/answers of normal quality in terms of fraction of best answers.

The distributions for the local clustering coefficient are shown in Figure 9. The separation in this case is better than with the number of triangles. In this case the Kolmogorov-Smirnov difference is 0.17 and the p -value for rejecting the null hypothesis is $7.9 \cdot 10^{-54}$. In general, the users in the set of high quality questions/answers have larger number of triangles and smaller local clustering coefficient.

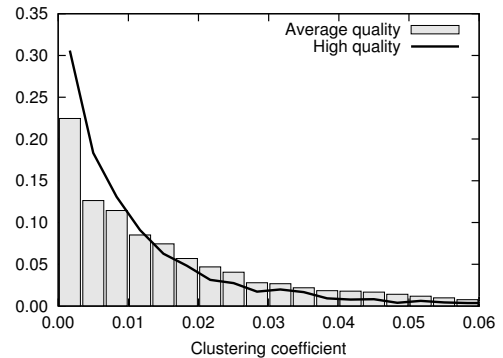


Figure 9: Separation of users who have provided questions/answers of high quality with users who have provided questions/answers of normal quality in terms of local clustering coefficient.

Notice that the partitioning of users into the sets H and N might not be very accurate since for each user there is usually only one question or answer that is evaluated. Thus, to obtain additional validation of our results we perform a second experiment, in which we partition the users into two sets: H_{ba} is the set of user who have fraction of best answers more than 30%, and N_{ba} is the set of the rest of the users. Then, as in the previous experiment, we examine if the distribution of the number of triangles and the distribution of the local clustering coefficient differ between the sets H_{ba} and N_{ba} . For the number of triangles, the Kolmogorov-Smirnov test rejects the null hypothesis with difference value equal to 0.11 and p -value equal to $4.5 \cdot 10^{-1}$. The separation is again more clear for the case of local clustering coefficient. The Kolmogorov-Smirnov difference is 0.27 and the p -value for rejecting the null hypothesis is $1.8 \cdot 10^{-59}$. We remark

that using only the degree of each user in the graph is not sufficient to distinguish between the two distributions.

8. CONCLUSIONS

We have presented efficient semi-streaming algorithms for counting the local number of triangles in a large graph. To the best of our knowledge, these are the first such algorithms described in the literature. We believe that there are many applications for such algorithms to Web-scale problems, and we have demonstrated two such applications.

For future work, exploring variants of the first algorithm that relax the semi-streaming constraint but still use a small amount of memory is promising. Given that the distribution of the number of triangles is very skewed, the counters Z_{uv} could be compressed. For instance, if the counters follow a power-law, a suitable coding could be used to store them. Note that each counter will use a variable number of bits depending on the value being stored. This may cause a drop in performance if done in external memory, but could be a good choice if done in main memory.

Data and code. The data graphs we used in this paper can be freely downloaded from <http://webgraph.dsi.unimi.it/>; the graph from Yahoo! Answers cannot be released publicly for privacy reasons. The Java code used for computing all the estimations, implementing the algorithms we have described, is freely available under a GPL license at <http://law.dsi.unimi.it/satellite-software/>.

Acknowledgments: we thank Massimo Santini and Sebastiano Vigna for valuable comments and feedback about a preliminary version of this work.

9. REFERENCES

- [1] N. Alon, R. Yuster, and U. Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997.
- [2] Z. Bar-Yossef, R. Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *SODA*, 2002.
- [3] V. Batagelj and A. Mrvar. A subquadratic triad census algorithm for large sparse networks with small maximum degree. *Social Networks*, 23:237–243, 2001.
- [4] T. Bohman, C. Cooper, and A. M. Frieze. Min-wise independent linear permutations. *Electr. J. Comb*, 7, 2000.
- [5] P. Boldi and S. Vigna. The webgraph framework I: compression techniques. In *WWW*, 2004.
- [6] A. Z. Broder. On the resemblance and containment of documents. In *Compression and Complexity of Sequences*, IEEE Computer Society, 1998.
- [7] A. Z. Broder. Identifying and filtering near-duplicate documents. In *CPM*. Springer, 2000.
- [8] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-wise independent permutations. In *STOC*, New York, NY, USA, 1998.
- [9] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the web. In *WWW*, 1997.
- [10] L. S. Buriol, G. Frahling, S. Leonardi, A. Marchetti-Spaccamela, and C. Sohler. Counting triangles in data streams. In *PODS*, 2006.
- [11] C. Castillo, D. Donato, L. Becchetti, P. Boldi, S. Leonardi, M. Santini, and S. Vigna. A reference collection for web spam. *SIGIR Forum*, 40(2):11–24, December 2006.
- [12] C. Castillo, D. Donato, A. Gionis, V. Murdock, and F. Silvestri. Know your neighbors: Web spam detection using the web topology. In *SIGIR*, 2007.
- [13] D. Coppersmith and R. Kumar. An improved data stream algorithm for frequency moments. In *SODA*, 2004.
- [14] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251–280, 1990.
- [15] C. Demetrescu, I. Finocchi, and A. Ribichini. Trading off space for passes in graph streaming problems. In *SODA*, 2006.
- [16] J.-P. Eckmann and E. Moses. Curvature of co-links uncovers hidden thematic layers in the world wide web. *PNAS*, 99(9):5825–5829, 2002.
- [17] J. Feigenbaum, S. Kannan, M. A. Gregor, S. Suri, and J. Zhang. On graph problems in a semi-streaming model. In *ICALP*, 2004.
- [18] D. Fetterly, M. Manasse, and M. Najork. Spam, damn spam, and statistics: Using statistical analysis to locate spam web pages. In *WebDB*, 2004.
- [19] D. Fogaras and B. Rácz. Scaling link-based similarity search. In *WWW*, 2005.
- [20] D. Gibson, R. Kumar, and A. Tomkins. Discovering large dense subgraphs in massive graphs. In *VLDB*, 2005.
- [21] A. Gulli and A. Signorini. The indexable Web is more than 11.5 billion pages. In *WWW*, 2005.
- [22] T. Haveliwala. Efficient computation of pagerank. Technical report, Stanford University, 1999.
- [23] M. R. Henzinger, P. Raghavan, and S. Rajagopalan. Computing on data streams. *Dimacs Series In Discrete Mathematics And Theoretical Computer Science*, pages 107–118, 1999.
- [24] P. Indyk. A small approximately min-wise independent family of hash functions. In *SODA*, 1999.
- [25] A. Itai and M. Rodeh. Finding a minimum circuit in a graph. *SIAM Journal of Computing*, 7(4):413–423, 1978.
- [26] G. Jeh and J. Widom. Simrank: a measure of structural-context similarity. In *KDD*, New York, NY, USA, 2002.
- [27] R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Trawling the Web for emerging cyber-communities. *Computer Networks*, 31(11–16):1481–1493, 1999.
- [28] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *KDD*, 2005.
- [29] M. E. J. Newman. The structure and function of complex networks. *SIAM Review*, 45(2):167–256, 2003.
- [30] T. Schank and D. Wagner. Finding, counting and listing all triangles in large graphs, an experimental study. In *Proceedings of the 4th International Workshop on Experimental and Efficient Algorithms (WEA)*, 2005.
- [31] J. S. Vitter. External memory algorithms and data structures. *ACM Computing Surveys*, 33(2):209–271, 2001.
- [32] H. T. Welsch, E. Gleave, D. Fisher, and M. Smith. Visualizing the signatures of social roles in online discussion groups. *The Journal of Social Structure*, 8(2), 2007.