Efficient Algorithms for Large-Scale Local Triangle Counting

LUCA BECCHETTI
"Sapienza" Università di Roma, Rome, Italy, Luca.Becchetti@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
and
ARISTIDES GIONIS, Yahoo! Research, Barcelona Spain, gionis@yahoo-inc.com

In this paper we study the problem of approximate 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. We consider the question both for undirected and directed graphs. The problem of computing the *global* number of triangles in a graph has been considered before, but to our knowledge this is the first contribution that addresses the problem of approximate *local* triangle counting with a focus on the efficiency issues arising in massive graphs and that also considers the directed case. 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 detect the presence of spamming activity in large-scale Web graphs, as well as to provide useful features for content quality assessment in social networks.

For computing the local number of triangles (undirected and directed) 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 of external memory during computation, while the second algorithm uses only main memory. We present the theoretical analysis as well as experimental results on large graphs, demonstrating the practical efficiency of our approach.

Categories and Subject Descriptors: H.3 [**Information Systems**]: Information Storage and Retrieval; I.5 [**Computing Methodologies**]: Pattern Recognition; G.1 [**Mathematics of Computing**]: Numerical Analysis; G.3 [**Mathematics of Computing**]: Probability and Statistics

General Terms: Algorithms; Experimentation; Performance
Additional Key Words and Phrases: Clustering Coefficient; Massive-Graph Computing; Web Computing; Social Networks

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 analyzing graph structures, and may reveal their underlying properties.

In this paper we study the problem of counting the local number of triangles in large graphs, both directed and undirected. In the undirected case, we consider 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 interested in computing, for every node u, the number of triangles incident to u, defined as:

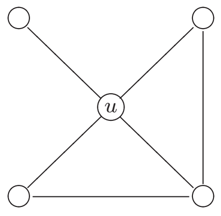
$$T(u) = \frac{1}{2}|\{e_{vw} \in E : e_{uv} \in E, e_{uw} \in E\}| \ .$$

Extending this definition to directed graphs is slightly more complicated, the first difficulty being the different possible ways of defining a directed triangle, which we defer to a further section of this paper.

The problem of counting triangles in undirected graphs also translates into computing the local clustering coefficient (also known as transitivity coefficient). For a node u, the local clustering coefficient is defined as

$$C(u) = \frac{2T(u)}{d_u(d_u - 1)} \ ,$$

that is, the ratio between the number of triangles and the largest possible number of triangles in which the node could participate. Figure 1 depicts an example.



$$\begin{aligned} d_u &= 4 \\ T(u) &= 2 \\ C(u) &= \frac{2 \cdot 2}{4 \cdot 3} = \frac{1}{3} \end{aligned}$$

Fig. 1. Example number of triangles and clustering coefficient for a node.

Note that the problem of estimating the *global* number of triangles in a graph or other structures such as other minors and bipartite cliques has been studied already, see e.g., [Bordino et al. 2008; Bar-Yossef et al. 2002; Buriol et al. 2006; Buriol et al. 2007]; here we deal with the problem of estimating simultaneously the local number of triangles of all the individual nodes in the graph.

Applications. 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 [Welser et al. 2007] (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, estimating the local number of triangles and local clustering coefficient can have a larger variety of other potential applications, ranging from the analysis of social or biological networks [Newman 2003] to the uncovering of thematic relationships in the Web [Eckmann and Moses 2002].

Algorithms. 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 [Broder 1998; Broder et al. 1998; Broder et al. 1997]. 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 [Broder 2000; Broder et al. 1998; Broder et al. 1997], 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 studying the role of collisions. A similar analysis had been sketched in [Broder 1998].

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 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.

Roadmap. 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 [Broder et al. 1998]. Section 5 presents our first algorithm, and Section 6 the main-memory-only algorithm. In Section 7 we describe experimental work assessing the effectiveness of the indices we consider in performing important mining tasks in the Web and social networks. The last section presents our conclusions and outlines future work.

2. RELATED WORK

Computing the clustering coefficient and the distribution of triangles is important to quantitatively assess the community structure of social networks [Newman 2003] or the thematic structure of large, hyperlinked document collections, such as the Web [Eckmann and Moses 2002].

There has been work on the exact computation of the number of triangles incident to each node in a graph [Alon et al. 1997; Batagelj and Mrvar 2001; Itai and Rodeh 1978]. 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 $\omega \leq 2.376$ [Coppersmith and Winograd 1990]. The algorithm based on matrix multiplication is currently the fastest main-memory algorithm for computing the number of triangles incident to each node. However, the space complexity of this algorithm is $O(|V|^2)$, which make it non practical for large graphs. The algorithm of Itai and Rodeh [Itai and Rodeh 1978] runs in time $O(|E|^{\frac{3}{2}})$ and requires $O(|E|)$ space, thus, it is slower in the general case but it has better space utilization, and it is preferable for sparse graphs.

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 "iterator" algorithms can be found in [Schank and Wagner 2005]; however, their running time is $O(|V|d^2_{\max})$ and $O(\sum_{v \in V} d_v^2)$, respectively. An excellent survey of main-memory algorithms for exact counting and triangle listing is provided by Lapaty [Latapy 2008]. In addition, Lapaty presents an improved analysis of exact triangle counting algorithms in the case that the input graph has a power-law degree distribution. However, the running time remains superlinear for all values of the power-law exponent α.

In summary, for large scale-free networks —having a very large number of nodes and high-degree nodes due to skewed degree distributions—exact computations are not scalable. Thus in this paper we resort to approximation algorithms.

In [Coppersmith and Kumar 2004] 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 [Buriol et al. 2006]. We remark that, differently from [Coppersmith and Kumar 2004; Buriol et al. 2006], 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 [Broder et al. 1997; Broder 1998; 2000]. Implementing min-wise independent permutations is infeasible in practice, since it would require exponential space [Broder et al. 1998]. In recent years, families of linear hash functions have been proposed that implement min-wise independent permutations approximately [Indyk 1999; Bohman et al. 2000]. 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 lead 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 [Gibson et al. 2005], the authors use the techniques of shingles and linear hashing to discover groups 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 [Kumar et al. 1999]. Finally, in [Fogaras and Rácz 2005], the authors apply similar techniques to produce indices of page similarity that extend SimRank [Jeh and Widom 2002].

In a preliminary version of this paper [Becchetti et al. 2008] we considered only undirected graphs, and we only sketched the theoretical analysis of our method. More recently, Tsourakakis [Tsourakakis 2008] described a completely different algorithm, based on computing the largest eigenvalues of the adjacency matrix of an undirected graph to approximate both the total as well as the local number of triangles in the graph; Tsourakis' method exploits an interesting property: the total number of triangles in an undirected graph is 1/6 of the sum of the cubes of the eigenvalues of its adjacency matrix.

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} [Gulli and Signorini 2005], 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 [Henzinger et al. 1999], 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 [Feigenbaum et al. 2004; Demetrescu et al. 2006]. This implies a semi-external memory constraint [Vitter 2001] 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 [Haveliwala 1999].

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 compute 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. The number of edges found in large scale-free graphs is typically small. However, despite the fact that $|E|$ is small compared to $|V|^2$, there is evidence that it may be $\omega(|V|)$ [Holme et al. 2004; Latapy and Magnien 2006; Leskovec et al. 2005].

3.2 Counting triangles in the undirected case

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 $e_{uv} \in E$, 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 [Broder et al. 1998; Broder et al. 1997; Broder 1998; 2000] to estimate the Jaccard coefficient and thus the size of the intersection of two sets A and B, defined over the same universe which we assume, without loss of generality, to be $[n] = \{0, \ldots, n-1\}$, where $n = 2^k$ for some suitable k.

3.3 Counting directed triangles

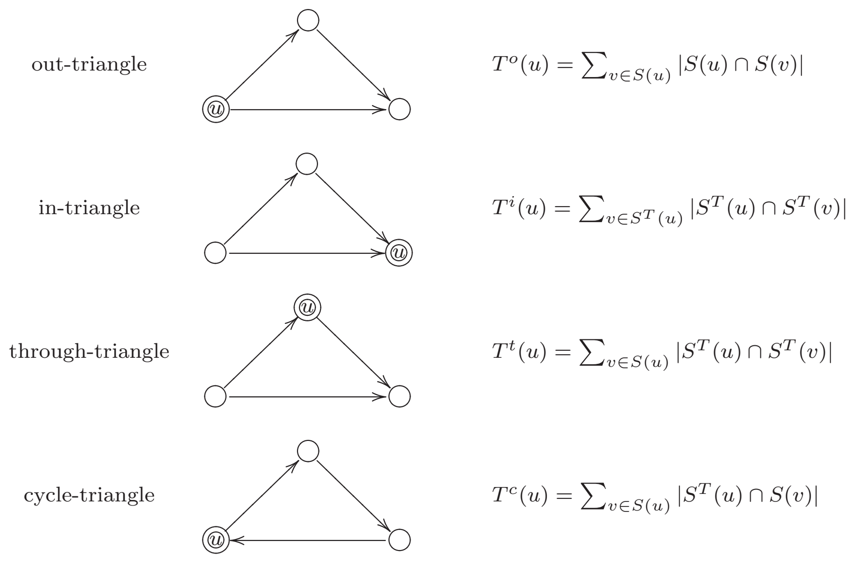
Even though the analysis we provide in the rest of this paper is referred to undirected graphs, the techniques we discuss may be applied (directly, or with minor modifications) to directed graphs. In this section we briefly discuss the issues that arise in the directed case.

The first problem is the very definition of directed triangles. Consider a directed graph $D = (N, A)$, and for every arc[1] $a = (x, y) \in A$, let $i(a) = \{x, y\}$ be the set of nodes on which the arc is incident. Three arcs a_1, a_2, a_3 are said to form a *(directed) triangle* [de Graaf et al. 1992] iff any two of them share a single endpoint (i.e., $|i(a_1) \cap i(a_2)| = |i(a_2) \cap i(a_3)| = |i(a_1) \cap i(a_3)| = 1$). In particular, all 3-cycles are directed triangles; a directed triangle that is not a 3-cycle is sometimes called a *transitive triangle*. Clearly, every directed triangle of D is a triangle in the undirected version of D, but the same undirected triangle may actually correspond to more than one directed triangle in D; for this reason, the definition in the directed case is given using arcs instead of nodes.

Since transitive triangles are rigid subgraphs, the three nodes involved cannot be interchanged (differently from 3-cycles, where all nodes are exchanged by some automorphism). As a result, a node u can be involved in a triangle in four possible ways; more precisely, given a triangle $T = \{a_1, a_2, a_3\}$ and a node $u \in i(a_1) \cup i(a_2) \cup i(a_3)$, we say that (see Figure 2)

—T is an *out-triangle* for u iff T is transitive and u has two outgoing arcs in T;

[1]Following the standard terminology, we use the words "node" and "arc" for directed graphs, and reserve "vertex" and "edge" for undirected graphs. All the directed graphs we consider are loopless (i.e., for all nodes $x \in N$ we have $(x, x) \notin A$).



$T^o(u) = \sum_{v \in S(u)} |S(u) \cap S(v)|$

$T^i(u) = \sum_{v \in S^T(u)} |S^T(u) \cap S^T(v)|$

$T^t(u) = \sum_{v \in S(u)} |S^T(u) \cap S^T(v)|$

$T^c(u) = \sum_{v \in S(u)} |S^T(u) \cap S(v)|$

Fig. 2. Directed triangles involving u.

—T is an *in-triangle* for u iff T is transitive and u has zero outgoing arcs in T;

—T is an *through-triangle* for u iff T is transitive and u has one outgoing arc in T;

—T is an *cycle-triangle* for u iff T is a 3-cycle.

We write $T^o(u)$ ($T^i(u)$, $T^t(u)$, $T^c(u)$, respectively) for the number of out- (in-, through-, cycle-, respectively) triangles involving u.

Figure 2 shows how the number of triangles of each type can be computed, based on the sets of out-neighbors $S(u) = \{v \mid (u, v) \in A\}$ and in-neighbors $S^T(u) = \{v \mid (v, u) \in A\}$ of u in the directed graph. Even if in the rest of the paper we will mainly be dealing with undirected graphs, we will get back to these formulas from time to time to explain how our algorithms can be modified to work in the directed case.

3.4 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/). Loops were not considered, and we ran our experiments both on the original directed graphs, as well as on symmetrized versions of them when testing our algorithms for the undirected case. We used the WebGraph framework [Boldi and Vigna 2004] to manipulate the graphs in compressed form. The particular collections we used are listed in Table I. 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 [Leskovec et al. 2005], 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 [Castillo et al. 2006] so we have labels of non-spam/spam for a large set of hosts in that collection.

Table I. 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

The distribution of the exact number of triangles per node in the smaller graph `EU-2005` is shown in Figure 3 and follows a power law.

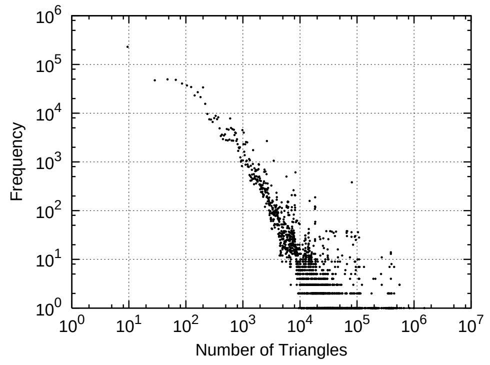


Fig. 3. Distribution of the number of triangles per node in the EU-2005 graph.

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 nodes 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. In other words, we know a set of users who contribute high-quality content to Yahoo! Answers.

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. [Broder 1998; 2000; Broder et al. 1998; Broder et al. 1997]. Assume we are able to choose uniformly at random a permutation $\pi(\cdot)$ mapping $[n]$ onto itself. 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 its minimum. Then it can be shown [Broder 2000] that (*i*) for every $a \in A \subseteq [n]$, $\mathbf{Pr}[a = \arg\min(\pi(A))] = 1/|A|$; (*ii*) for every A, $B \subseteq [n]$: $\mathbf{Pr}[\min(\pi(A)) = \min(\pi(B))] = J(A, B)$. 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 [Broder et al. 1998]. In practice, suitable families of linear hash functions are used (e.g. see [Indyk 1999; Bohman et al. 2000]).

In this paper, in order to increase the speed of computation, we adopt a slight modification of this approach, using a pseudo-random number generator to assign labels to the graph's vertices. In particular, we show that as long as the pseudo-random number generator reproduces a uniform distribution closely enough, collisions are not too frequent and it is possible to approximate the Jaccard coefficient satisfactorily. Very efficient pseudo-random generator algorithms have been proposed in the literature, exhibiting excellent statistical properties. In practice, we used the Mersenne Twister, which is a fast generation algorithm for obtaining high-quality pseudo-random numbers.

Figure 4 describes the algorithm's pseudo-code, which is exactly the standard one given for example in [Broder 2000], except for the use of random labels. As for 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 $i : 1 \ldots m$ 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 $\leftarrow$ (count/(count + $m$))($|A| + |B|$)
```

Fig. 4. Basic algorithm for estimating the size 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 set $W = \sum_{i=1}^m W_i$. Set $X = |A \cap B|$. Our estimator of X is $\widehat{X} = W/(W+m)(|A|+|B|)$. 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 [Broder et al. 1998]. 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 [Broder et al. 1997; Broder 1998; 2000]. We present this result here for the sake of completeness, since it considers the role of collisions (an aspect only sketched in [Broder 1998]).

THEOREM 4.1. *For every $\epsilon > 0$ and for every number m of iterations:*

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

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 Section 5, we will describe how to apply the same techniques for estimating the number of triangles.

4.1 Proof of Theorem 4.1

In order to prove Theorem 4.1, let $[n] = \{0, \ldots, n-1\}$, and define a *semi-permutation* of $[n]$ as a map $h : [n] \to [n]$. Notice that we do not require injectivity: there might exist $i \neq j$ such that $h(i) = h(j)$.

For every $A \subseteq [n]$ and $i = 1, \ldots, m$, denote by $\mathbf{L}^i(A)$ the minimum value of labels assigned to elements in A during the i-th iteration of the algorithm and by $\mathcal{M}^i(A)$ the set of elements achieving $\mathbf{L}^i(A)$. We have the following

LEMMA 4.2. *Assume that during the i-th iteration each element in the set $[n]$ receives a label drawn uniformly at random between 0 and $2^k - 1$. For all $A, B \subseteq [n]$ we have*

$$\mathbf{Pr}\big[\mathbf{L}^i(A) = \mathbf{L}^i(B) \mid |\mathcal{M}^i(A \cup B)| = 1\big] = J(A, B).$$

PROOF. Notice that, if $|\mathcal{M}^i(A \cup B)| = 1$, then $(\mathbf{L}^i(A) = \mathbf{L}^i(B))$ if and only if the minimum is achieved by some element in $A \cap B$. Assuming an initial order $\{a_1, \ldots, a_{|A \cup B|}\}$ for elements in $A \cup B$ and recalling that labelling is described by a map h, every semi-permutation is described by a vector of size $|A \cup B|$, its i-th element being $h(a_i)$. For every element in $a_j \in A \cup B$, define by $\Psi(a_j)$ the set of semi-permutations such that i) $a_j \in \mathcal{M}^i(A \cup B)$ and ii) $|\mathcal{M}^i(A \cup B)| = 1$. For every $a_j, a_l \in A \cup B$, $j \neq l$, we have $|\Psi(a_j)| = |\Psi(a_l)| = P$ for some value P, by symmetry[2]. Also, note that $\Psi(a_j) \cap \Psi(a_l) = \emptyset$, for every $a_j, a_l \in A \cup B$, $j \neq l$. Hence, the overall number of semi-permutations such that there is one minimum and this minimum is achieved by an element in $A \cap B$ is $|A \cap B|P$. By the same argument, $|A \cup B|P$ is the number of semi-permutations, such that $(|\mathcal{M}^i(A \cup B)| = 1)$. Finally, since semi-permutations are chosen uniformly at random:

$$\mathbf{Pr}\big[\mathbf{L}^i(A) = \mathbf{L}^i(B) \mid |\mathcal{M}^i(A \cup B)| = 1\big] = \frac{|A \cap B|P}{|A \cup B|P} = J(A, B).$$

□

In the sequel we will use the following, well known fact:

FACT 4.3. *For every integer $n > 1$:*

$$\left(1 - \frac{1}{n}\right)^n < \frac{1}{e} < \left(1 - \frac{1}{n+1}\right)^n.$$

Recall that $W_i = 1$ if and only if, in the i-th iteration, $\mathbf{L}^i(A) = \mathbf{L}^i(B)$ and $W = \sum_{i=1}^m W_i$. Define $E_i = 1$ if $(|\mathcal{M}^i(A \cup B)| > 1)$, 0 otherwise and let $E = \sum_{i=1}^m E_i$.

LEMMA 4.4.

$$\mathbf{E}[W \mid E = 0] = mJ(A, B).$$

[2] The exact value of P is irrelevant to the analysis.

Proof. We have:

$$\begin{aligned}\mathbf{E}[W \mid E = 0] &= \mathbf{E}\big[W \mid \cap_{j=1}^{m} E_j = 0\big]\\ &= \sum_{i=1}^{m} \mathbf{Pr}\big[W_i = 1 \mid \cap_{j=1}^{m} E_j = 0\big]\\ &= \sum_{i=1}^{m} \mathbf{Pr}[W_i = 1 \mid E_i = 0] = mJ(A, B),\end{aligned}$$

where the second equality follows since W_i is independent of what happens in iterations different from the i-th one. □

Lemma 4.5.

$$\mathbf{Pr}[E > 0] < \frac{m|A \cup B|}{2^k - 1}.$$

Proof. We set $s = |A \cup B|$ for the rest of this proof. Consider the generic i-th iteration. We have:

$$\begin{aligned}\mathbf{Pr}[E_i = 0] &= \mathbf{Pr}\big[|\mathcal{M}^i(A \cup B)| = 1\big]\\ &= |A \cup B| \sum_{x=0}^{2^k-1} \frac{1}{2^k}\left(1 - \frac{x+1}{2^k}\right)^{s-1}.\end{aligned}$$

In deriving the expression above, for every element $a_j \in A \cup B$ and for every possible value x, we computed the probability that a_j receives the value x and all other elements receive higher labels. We further have:

$$\begin{aligned}\sum_{x=0}^{2^k-1}\left(1 - \frac{x+1}{2^k}\right)^{s-1} &\geq \int_0^{2^k-1}\left(1 - \frac{x+1}{2^k}\right)^{s-1} dx\\ &= \frac{2^k}{s}\left(1 - \frac{1}{2^k}\right)^{s} > e^{-\frac{s}{2^k-1}}.\end{aligned}$$

Here, the last inequality follows from Fact 4.3 after simple manipulations, observing that $2^k \geq s$. Hence we have:

$$\begin{aligned}\mathbf{Pr}[E_i = 1] &= 1 - \mathbf{Pr}\big[|\mathcal{M}^i(A \cup B)| = 1\big] < 1 - e^{-\frac{s}{2^k-1}}\\ &< \frac{s}{2^k - 1},\end{aligned}$$

where the last inequality follows since $e^{-x} > 1 - x$. This implies the lemma. □

Remark. Note that according to Lemma 4.5, we can make the probability of collisions occurring arbitrarily small by choosing k sufficiently large. More precisely, if A and B belong to a universe of size n, it is enough to choose $k = \Omega(\log n + \log m)$. In particular, when estimating the number of triangles in a graph, A and B will be the neighbourhoods of any two vertices of the graph, while n will be the number of vertices in the graph.

Next, set $X = |A \cap B|$. Our estimator of X is $\widehat{X} = W/(W + m)(|A| + |B|)$. The next result states that, up to collisions, our estimation of $|A \cap B|$ differs from the true value by more than a constant with a probability that exponentially decays with m and $J(A,B)$.

Now observe that, by definition of the Jaccard coefficient, we have $J(A,B) = X/|A \cup B| = X/(|A| + |B| - X)$, which implies:

$$X = \frac{J(A,B)}{J(A,B)+1}(|A| + |B|).$$

Furthermore we have:

$$\begin{aligned}
&\mathbf{Pr}\Big[|\widehat{X} - X| > \epsilon X\Big] \\
&= \mathbf{Pr}\Big[(|\widehat{X} - X| > \epsilon X) \,|\, (E = 0)\Big]\,\mathbf{Pr}[E = 0] \\
&+\mathbf{Pr}\Big[(|\widehat{X} - X| > \epsilon X) \,|\, (E > 0)\Big]\,\mathbf{Pr}[E > 0] \\
&\leq \mathbf{Pr}\Big[(|\widehat{X} - X| > \epsilon X) \,|\, (E = 0)\Big] + \mathbf{Pr}[E > 0] \\
&< \mathbf{Pr}\Big[(|\widehat{X} - X| > \epsilon X) \,|\, (E = 0)\Big] + \frac{m|A \cup B|}{2^k - 1},
\end{aligned}$$

where the last inequality follows from Lemma 4.5. Now, for $i = 1, \ldots, m$, set $\overline{W}(i) = (W_i \,|\, E = 0)$ and $\overline{W} = \sum_{i=1}^{m} \overline{W}(i)$. Also observe that, by Lemma 4.4, $\mathbf{E}\big[\overline{W}\big] = mJ(A,B)$. Then, for a given realization $\overline{W}$, our "ideal" estimator $\overline{X}$ of X is:

$$\overline{X} = (\widehat{X} \,|\, E = 0) = \frac{\overline{W}}{\overline{W} + m}(|A| + |B|).$$

Remark. Note that by their definitions, $\overline{W}(i)$, $\overline{W}$ and $\overline{X}$ are no longer defined over the original probability space, but over the restricted probability space that is conditioned to the event $(E = 0)$.

Set $\overline{J} = \sum_{i=1}^{m} \overline{W}(i)/m = \overline{W}/m$. By the expression of X derived before and by the definition of $\overline{X}$ we have that $\overline{X} > (1+\epsilon)X$ implies

$$\frac{\overline{J}}{\overline{J} + 1} > (1+\epsilon)\frac{J(A,B)}{J(A,B)+1},$$

which in turn implies

$$\overline{J} > \frac{(1+\epsilon)J(A,B)}{1 - \epsilon J(A,B)} > (1+\epsilon)J(A,B),$$

that is:

$$\overline{W} > (1+\epsilon)mJ(A,B) = (1+\epsilon)\mathbf{E}\big[\overline{W}\big].$$

Analogously, $\overline{X} < (1-\epsilon)X$ implies:

$$\overline{J} < \frac{(1-\epsilon)J(A,B)}{1 + \epsilon J(A,B)} < (1-\epsilon)J(A,B),$$

and this in turn implies

$$\overline{W} < (1-\epsilon)mJ(A,B) = (1-\epsilon)\mathbf{E}\big[\overline{W}\big].$$

So we have:

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

Finally, the $\overline{W}(i)$'s are statistically independent. This clearly follows since $\overline{W}(i)$ only depends on E_i and not on E_j, for $j \neq i$. Hence, applying Chernoff bound [Mitzenmacher and Upfal 2005] to $\overline{W}$, we have:

$$\mathbf{Pr}\left[|\overline{W} - \mathbf{E}\left[\overline{W}\right] | > \epsilon \mathbf{E}\left[\overline{W}\right]\right] \leq 2e^{-\frac{\epsilon^2}{3} mJ(A,B)}.$$

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 $\widehat{T}(u)$ of the number of triangles $T(u)$ for all vertices in the graph.

5.1 An Algorithm for Undirected Graphs

The algorithm for computing the number of triangles is written in pseudo-code in Figure 5 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 assigned to node u at the p-th pass.

```
Require: directed graph $D = (N, A)$, 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 $u : 1 \dots |N|$ do {Initialize node labels and min}
 4:     $h_p(u) \leftarrow k$ random bits
 5:     $\min(u) \leftarrow +\infty$
 6:   for $src : 1 \dots |N|$ do {Compute minima}
 7:     for all links from $src$ to $dest$ do
 8:       $\min(src) \leftarrow \min(\min(src), h_p(dest))$
 9:   for $src : 1 \dots |N|$ 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 $src : 1 \dots |N|$ do {Compute number of triangles}
14:   $\widehat{T}(src) \leftarrow 0$
15:   for all links from $src$ to $dest$ do
16:     $\widehat{T}(src) \leftarrow \widehat{T}(src) + \frac{Z_{src,dest}}{Z_{src,dest}+m}(|S(src)| + |S(dest)|)$
17:   $\widehat{T}(src) \leftarrow \widehat{T}(src)/2$
18: return $\widehat{T}(\cdot)$
```

Fig. 5. 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 $Z_{\cdot,\cdot}$ for each edge is increased.

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

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

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 4.1, 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 pairs with higher Jaccard coefficient the error is smaller.

Remark. The value of m depends on the desired per-node accuracy. As Theorem 4.1 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 blockwise, 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:

$$\frac{1}{|V|} \sum_{u} \frac{|T(u) - \widehat{T}(u)|}{T(u)} .$$

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. This

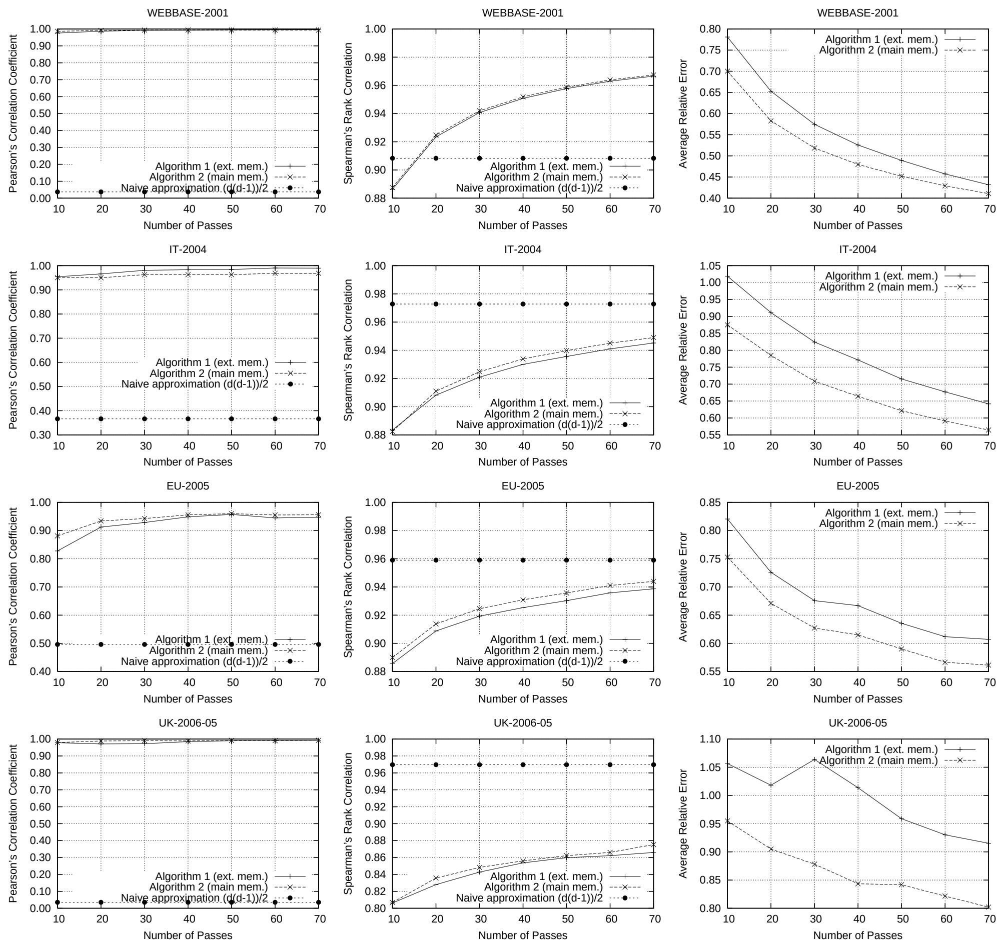


Fig. 6. Accuracy of the approximation of the number of triangles using the two algorithms described in the paper (external memory and main memory). Left: Pearson's correlation coefficient. Center: Spearman's rank correlation coefficient. Right: average relative error.

behaviour is summarized in Figure 6, where the error of these approximations is shown for all Web graphs of Table I.

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 in the Web samples considered follows a power law, as shown in Figure 3. A similar observation was also made in [Eckmann and Moses 2002], although for Web samples of smaller size.

5.3 The case of directed graphs

As we have seen, the correctness of the algorithm in Figure 5 depends on the fact that

$$T(v) = \frac{1}{2} \sum_{v \in S(u)} |S(u) \cap S(v)|$$

that is further approximated as

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

where Z_{uv} is the number of times the minimum label for $S(u)$ coincided with the minimum label for $S(v)$: the latter approximation step is actually an application of the algorithm in Figure 4.

Looking at the formulas in Figure 2, we can immediately see how the algorithm can be applied to the directed case with minor modifications. The modified algorithm is presented in Figures 7 and 8, and depends on the type of triangles one wants to count. In all cases, we need to estimate the intersection of two sets: either $|S(u) \cap S(v)|$, or $|S^T(u) \cap S^T(v)|$, or $|S^T(u) \cap S(v)|$:

—in the first case, we need to determine how often the label assigned to out-neighbors of u and v coincide (and the minimum label of the out-neighbors of every node x is computed in $\min^+(x)$);

—in the second case, instead, we need to see how often the label assigned to in-neighbors of u and v coincide (and the minimum label of in-neighbors of x is computed in $\min^-(x)$);

—in the last case, we need to compare the minimum label of in-neighbors of u with the minimum label of out-neighbors of v, so we will have to compute both $\min^+(x)$ and $\min^-(x)$ for every node x.

Note that the last case (i.e., when computing $T^c(-)$) is the only one requiring extra memory (we need to store two arrays of minima, instead of one). Moreover, except when computing $T^o(-)$, one always needs to know the in-degrees of nodes ($|S^T(x)|$, for every node x); depending on the way the graph is stored, we may have this information available at no cost, or we may need to pre-compute it at the beginning of the algorithm, with a single linear pass on the graph (and using $O(|V| \log |V|)$ extra bits of memory).

Accuracy. Note that the results of Theorem 4.1 apply in this case as well, where now A (B) can be either $S(u)$ or $S^T(u)$ (respectively, $S(v)$ or $S^T(v)$).

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. To this aim, observe that, in the final step

```
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:     $\boxed{\text{INITIALIZE MIN}}$
 6:   for $src : 1 \dots |V|$ do {Compute minima}
 7:     for all links from $src$ to $dest$ do
 8:       $\boxed{\text{COMPUTE MIN}}$
 9:   for $src : 1 \dots |V|$ do {Compare minima}
10:     for all links from $src$ to $dest$ do
11:       if $\boxed{\text{COMPARE MIN}}$ then
12:         $Z_{src,dest} \leftarrow Z_{src,dest} + 1$
13: for $src : 1 \dots |V|$ do {Compute number of triangles}
14:   $\widehat{T}(src) \leftarrow 0$
15:   for all links from $src$ to $dest$ do
16:     $\boxed{\text{COMPUTE SUM}}$
17: return $\widehat{T}(\cdot)$
```

Fig. 7. Algorithm for estimating the number of triangles of each node, in the case of directed graphs; some parts of the algorithm depend on the kind of triangles you want to compute (see Fig. 8). The counters $Z_{\cdot,\cdot}$ are kept on external memory and updated sequentially.

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

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

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:

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

The algorithm that uses this approximation is given in Figure 9 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 5, but removing Z_{uv} from the denominator in the expression of $\widehat{\widehat{T}}(u)$ 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 $|S(src)| + |S(dest)|$.

INITIALIZE MIN	out-triangles	$\min^+(u) \leftarrow +\infty$
	in- and through-triangles	$\min^-(u) \leftarrow +\infty$
	cycle-triangles	$\min^+(u) \leftarrow +\infty$ $\min^-(u) \leftarrow +\infty$
COMPUTE MIN	out-triangles	$\min^+(src) \leftarrow \min(\min^+(src), h_p(dest))$
	in- and through-triangles	$\min^-(dest) \leftarrow \min(\min^-(dest), h_p(src))$
	cycle-triangles	$\min^+(src) \leftarrow \min(\min^+(src), h_p(dest))$ $\min^-(dest) \leftarrow \min(\min^-(dest), h_p(src))$
COMPARE MIN	out-triangles	$\min^+(src) == \min^+(dest)$
	in- and through-triangles	$\min^-(src) == \min^-(dest)$
	cycle-triangles	$\min^-(src) == \min^+(dest)$
COMPUTE SUM	out-triangles	$\widehat{T}(src) \leftarrow \widehat{T}(src) + \frac{Z_{src,dest}}{Z_{src,dest}+m}(\lvert S(src)\rvert + \lvert S(dest)\rvert)$
	in-triangles	$\widehat{T}(dest) \leftarrow \widehat{T}(dest) + \frac{Z_{src,dest}}{Z_{src,dest}+m}(\lvert S^T(src)\rvert + \lvert S^T(dest)\rvert)$
	through-triangles	$\widehat{T}(src) \leftarrow \widehat{T}(src) + \frac{Z_{src,dest}}{Z_{src,dest}+m}(\lvert S^T(src)\rvert + \lvert S^T(dest)\rvert)$
	cycle-triangles	$\widehat{T}(src) \leftarrow \widehat{T}(src) + \frac{Z_{src,dest}}{Z_{src,dest}+m}(\lvert S^T(src)\rvert + \lvert S(dest)\rvert)$

Fig. 8. Variant parts of the algorithm in Fig. 7, depending on the kind of triangles you want to compute.

```
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:   $\widehat{\widehat{T}}(src) \leftarrow \frac{1}{3m} Z_u$
15: return $\widehat{\widehat{T}}(\cdot)$
```

Fig. 9. Algorithm for estimating the number of triangles of each node in main memory.

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

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

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.

6.1 Analysis

We can give a result similar to that of Theorem 4.1. 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 11 of the algorithm of Figure 9 is true for nodes u and v. Finally, define

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

We have

THEOREM 6.1.

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

PROOF. For $i = 1, \ldots, m$, 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:

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

Set $\overline{X} = (\widehat{X} \,|\, E = 0)$[3]. By the definitions of $\widehat{X}$ and $\overline{X}$ we have $\mathbf{E}\left[\overline{X}\right] = \frac{\mathbf{E}[\overline{W}]}{1.5m}(|S(u)| + |S(v)|)$. Furthermore, since $0 \leq \mathbf{E}\left[\overline{W}\right] \leq m$, we can conclude that we have:

$$\frac{2}{3}X \leq \mathbf{E}\left[\overline{X}\right] \leq \frac{4}{3}X.$$

We have:

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

[3]Note that, like in the proof of Theorem 4.1, the variables $\overline{W}(i)$, $\overline{W}$ and $\overline{X}$ are no longer defined over the original probability space, but over the restricted probability space that is conditioned to the event $(E = 0)$.

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

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

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

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

The rest of the proof now proceeds exactly as in Theorem 4.1, recalling that $\mathbf{E}\left[\overline{W}\right] = mJ(S(u), S(v))$. □

Remark. As to the choice of m, considerations analogous to those at the end of Subsection 5.1 hold. Moreover, we can adapt also this algorithm to work for directed graphs, using exactly the same techniques described in Section 5.3.

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 on 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 6 depicts the results for the case of IT-2004 (experiments on the other datasets have been omitted, but have essentialy the same behavior).

In the implementation, the number of bits necessary to store each counter depends on the number of iterations and on the average degree 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. In practice we observed that 64 bits were required 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 [Fetterly et al. 2004]. In this section we test if the number of triangles is a relevant feature for web spam detection. It has been shown in the past to be relevant for e-mail spam detection in personal e-mail networks [Boykin and Roychowdhury 2005].

We used the `WEBSPAM-UK2006` spam collection [Castillo et al. 2006], 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 10.

The distributions are different for non-spam and spam hosts. A two-tailed Kolmogorov-Smirnov test indicates that the number of undirected triangles and the clustering coefficient have distributions that are substantially different in the two 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 [Castillo et al. 2007]. We sorted all the features by computing the χ^2-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 11. Also in this case, a two-tailed Kolmogorov-Smirnov test 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 $1.54 \cdot 10^{-7}$.

7.2 Content quality in social networks

In [Welser et al. 2007] 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.

We consider an undirected graph $G = (V, E)$, where V is a set of users in the system, and an edge $e_{uv} \in E$ 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

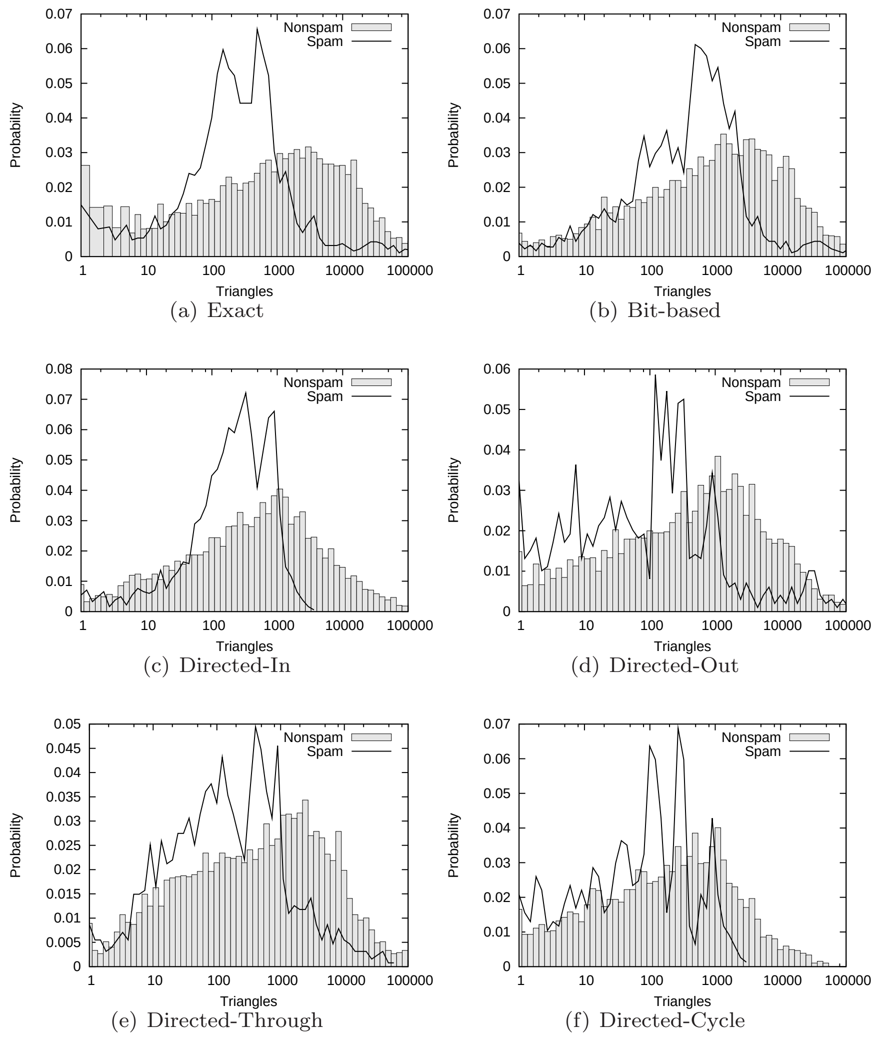


Fig. 10. Separation of non-spam and spam hosts in the histogram of triangles, measured using different algorithms. Approximate methods used 50 passes.

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 12, 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

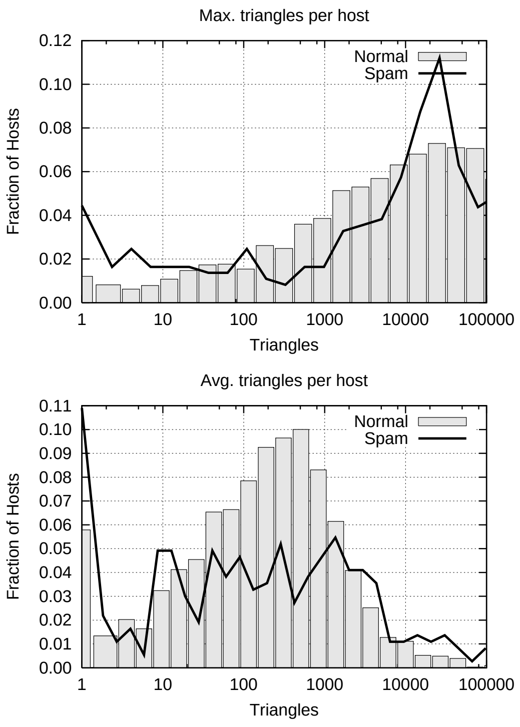


Fig. 11. Separation of non-spam and spam hosts in the histogram of triangles computed at page level and maximized/averaged on each host.

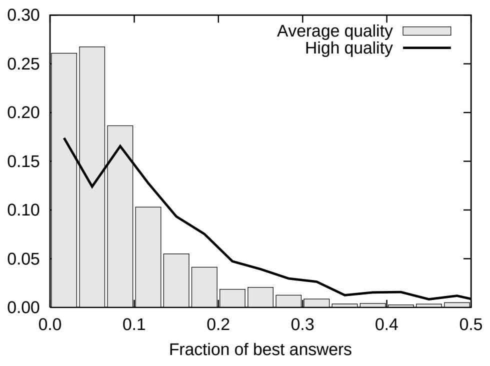


Fig. 12. Separation of users who have provided questions/answers of high quality with users who have provided questions/answers of normal quality in terms of fraction of best answers.

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 $2.9 \cdot 10^{-29}$.

The distributions for the local clustering coefficient are shown in Figure 13. 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.

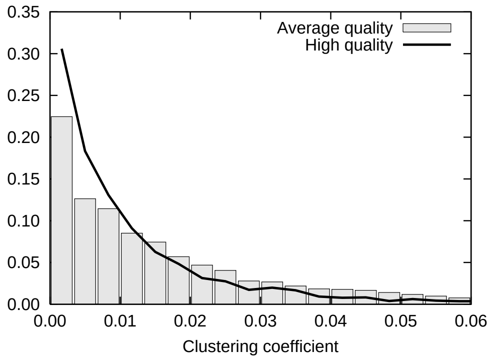


Fig. 13. Separation of users who have provided questions/answers of high quality with users who have provided questions/answers of normal quality in terms of local clustering coefficient.

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 Web-scale problems in which it is known that the local clustering coefficient plays a role, but that so far have not fully exploited it because of computational requirements. We have demonstrated that our approximate method can be useful for 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: 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 PRIN projects "Mathematical aspects and forthcoming applications of automata and formal languages", by the EU Integrated Project DELIS and by a Yahoo! Faculty Grant.

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

REFERENCES

Alon, N., Yuster, R., and Zwick, U. 1997. Finding and counting given length cycles. *Algorithmica 17,* 3, 209–223.

Bar-Yossef, Z., Kumar, R., and Sivakumar, D. 2002. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *Proceedings of the 13th Annual ACM-SIAM Symposium On Discrete Mathematics (SODA)*. 623–632.

Batagelj, V. and Mrvar, A. 2001. A subquadratic triad census algorithm for large sparse networks with small maximum degree. *Social Networks 23*, 237–243.

Becchetti, L., Boldi, P., Castillo, C., and Gionis, A. 2008. Efficient semi-streaming algorithms for local triangle counting in massive graphs. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. 16–24.

Bohman, T., Cooper, C., and Frieze, A. M. 2000. Min-wise independent linear permutations. *The Electronic Journal of Combinatorics, 7*.

Boldi, P. and Vigna, S. 2004. The webgraph framework I: compression techniques. In *Proceedings of the 13th international conference on World Wide Web*. 595–602.

Bordino, I., Donato, D., Gionis, A., and Leonardi, S. 2008. Mining large networks with subgraph counting. In *IEEE International Conference on Data Mining (ICDM)*. 737–742.

Boykin, P. O. and Roychowdhury, V. P. 2005. Leveraging social networks to fight spam. *Computer 38,* 4, 61–68.

Broder, A. Z. 1998. On the resemblance and containment of documents. In *Compression and Complexity of Sequences, IEEE Computer Society*. 21–29.

Broder, A. Z. 2000. Identifying and filtering near-duplicate documents. In *11th Annual Symposium on Combinatorial Pattern Matching (CPM)*. 1–10.

Broder, A. Z., Charikar, M., Frieze, A. M., and Mitzenmacher, M. 1998. Min-wise independent permutations (extended abstract). In *Proceedings of the thirtieth annual ACM symposium on Theory of computing (STOC)*. 327–336.

Broder, A. Z., Glassman, S. C., Manasse, M. S., and Zweig, G. 1997. Syntactic clustering of the web. In *6th International Conference on World Wide Web*. Elsevier Science Publishers Ltd., 1157–1166.

Buriol, L. S., Frahling, G., Leonardi, S., Marchetti-Spaccamela, A., and Sohler, C. 2006. Counting triangles in data streams. In *Proceedings of the 25th ACM Symposium on Principles of Database Systems (PODS)*. 253–262.

Buriol, L. S., Frahling, G., Leonardi, S., and Sohler, C. 2007. Estimating clustering indexes in data streams. In *15th Annual European Symposium on Algorithms*. 618–632.

Castillo, C., Donato, D., Becchetti, L., Boldi, P., Leonardi, S., Santini, M., and Vigna, S. 2006. A reference collection for web spam. *SIGIR Forum 40,* 2, 11–24.

Castillo, C., Donato, D., Gionis, A., Murdock, V., and Silvestri, F. 2007. Know your neighbors: Web spam detection using the web topology. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. 423–430.

Coppersmith, D. and Kumar, R. 2004. An improved data stream algorithm for frequency moments. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 151–156.

Coppersmith, D. and Winograd, S. 1990. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation 9,* 3, 251–280.

de Graaf, M., Schrijver, A., and Seymour, P. D. 1992. Directed triangles in directed graphs. *Discrete Mathematics 110,* 1-3, 279–282.

Demetrescu, C., Finocchi, I., and Ribichini, A. 2006. Trading off space for passes in graph streaming problems. In *Proceedings of the 7th annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 714–723.

Eckmann, J.-P. and Moses, E. 2002. Curvature of co-links uncovers hidden thematic layers in the world wide web. *Proceedings of the National Academy of Sciences (PNAS) 99,* 9, 5825–5829.

Feigenbaum, J., Kannan, S., Gregor, M. A., Suri, S., and Zhang, J. 2004. On graph problems in a semi-streaming model. In *31st International Colloquium on Automata, Languages and Programming (ICALP)*. 207–216.

Fetterly, D., Manasse, M., and Najork, M. 2004. Spam, damn spam, and statistics: Using statistical analysis to locate spam web pages. In *Proceedings of the seventh workshop on the Web and databases (WebDB)*. 1–6.

Fogaras, D. and Rácz, B. 2005. Scaling link-based similarity search. In *Proceedings of the 14th International Conference on World Wide Web*. 641–650.

Gibson, D., Kumar, R., and Tomkins, A. 2005. Discovering large dense subgraphs in massive graphs. In *Proceedings of the 31st international conference on Very large data bases (VLDB)*. 721–732.

Gulli, A. and Signorini, A. 2005. The indexable Web is more than 11.5 billion pages. In *Poster proceedings of the 14th international conference on World Wide Web*. 902–903.

Haveliwala, T. 1999. Efficient computation of pagerank. Tech. rep., Stanford University.

Henzinger, M. R., Raghavan, P., and Rajagopalan, S. 1999. Computing on data streams. *Dimacs Series In Discrete Mathematics And Theoretical Computer Science*, 107–118.

Holme, P., Edling, C., and Liljeros, F. 2004. Structure and time-evolution of an internet dating community. *Social Networks 26*, 155.

Indyk, P. 1999. A small approximately min-wise independent family of hash functions. In *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 454–456.

Itai, A. and Rodeh, M. 1978. Finding a minimum circuit in a graph. *SIAM Journal of Computing 7,* 4, 413–423.

Jeh, G. and Widom, J. 2002. Simrank: a measure of structural-context similarity. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM Press, New York, NY, USA, 538–543.

Kumar, R., Raghavan, P., Rajagopalan, S., and Tomkins, A. 1999. Trawling the Web for emerging cyber-communities. *Computer Networks 31,* 11–16, 1481–1493.

Latapy, M. 2008. Main-memory triangle computations for very large (sparse (power-law)) graphs. *Theoretical Computer Science 407,* 1-3, 458–473.

Latapy, M. and Magnien, C. 2006. Measuring fundamental properties of real-world complex networks. *arXiv:cs/0609115v2*.

Leskovec, J., Kleinberg, J., and Faloutsos, C. 2005. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the 11th ACM SIGKDD international conference on Knowledge discovery in data mining.* 177–187.

Mitzenmacher, M. and Upfal, E. 2005. *Probability and Computing : Randomized Algorithms and Probabilistic Analysis.* Cambridge University Press.

Newman, M. E. J. 2003. The structure and function of complex networks. *SIAM Review 45,* 2, 167–256.

Schank, T. and Wagner, D. 2005. 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).*

Tsourakakis, C. E. 2008. Fast counting of triangles in large real networks without counting: Algorithms and laws. In *ICDM '08: Proceedings of the 2008 Eighth IEEE International Conference on Data Mining.* IEEE Computer Society, Washington, DC, USA, 608–617.

Vitter, J. S. 2001. External memory algorithms and data structures. *ACM Computing Surveys 33,* 2, 209–271.

Welser, H. T., Gleave, E., Fisher, D., and Smith, M. 2007. Visualizing the signatures of social roles in online discussion groups. *The Journal of Social Structure 8,* 2.