Efficient computation of the Weighted Clustering Coefficient

Silvio Lattanzi
Google Research NY
silviol@google.com

Stefano Leonardi [*]
Universitá di Roma,"La Sapienza"
leonardi@dis.uniroma1.it

Abstract

The clustering coefficient of an unweighted network has been extensively used to quantify how tightly connected is the neighbor around a node and it has been widely adopted for assessing the quality of nodes in a social network. The computation of the clustering coefficient is challenging since it requires to count the number of triangles in the graph. Several recent works proposed efficient sampling, streaming and MapReduce algorithms that allow to overcome this computational bottleneck.

As a matter of fact, the intensity of the interaction between nodes, that is usually represented with weights on the edges of the graph, is also an important measure of the statistical cohesiveness of a network. Recently various notions of weighted clustering coefficient have been proposed but all those techniques are hard to implement on large-scale graphs.

In this work we show how standard sampling techniques can be used to obtain efficient estimators for the most commonly used measures of weighted clustering coefficient. Furthermore we also propose a novel graph-theoretic notion of clustering coefficient in weighted networks.

1 Introduction

In recent years we observed a growing attention on the study of the structural properties of social networks [16, 18] as result of the fast increase of the amount of social network data available for research. A widely adopted measure of the graph structure of a social network is the clustering coefficient [34]. The local clustering coefficient of a node is defined as the probability that any two neighbors of a node are themselves neighbors. The clustering

[*] Part of this work has been done while visiting Google Research, NY. Partly supported from Google Focused award "Web Algorithmics for Large-scale Data Analysis" and ERC StG PAAI "Practical Approximation Algorithms".

coefficient of a graph is the average local clustering coefficient of the nodes of the graph.

The clustering coefficient is used to measure how tightly interconnected the community is around a node. The degree of closeness of any two neighbors of a node is also interpreted as an index of trust of the node itself. The local clustering coefficient of a node has been proved for example to be a relevant feature for detecting spam nodes in the web [3] and high quality users in social networks [3].

Computing the clustering coefficient of a network is a challenging computational task since it reduces to counting the number of triangles in a graph. This task can be naively executed in $O(n^3)$ time or it can be reduced to matrix multiplication. The problem of computing the local clustering coefficient for every node of the network is even more challenging. Several recent works have proposed a variety of efficient methods for fast computation of clustering coefficient in large scale networks based on random sampling [11], streaming algorithms [7, 14], and MapReduce parallel computation [30].

However, most of the studies on the structural properties of social networks have focused on unweighted networks. In practice, many real world networks exhibit a varying degree of intensity and heterogeneity in the connections which is usually modeled with positive real weights on edges. Weights on edges are used for instance to measure the number of messages exchanged between friends or the number of links between hosts. Since the statistical level of cohesiveness in a network should in principle depend also on the weight of the edges, some recent interesting papers started to investigate weighted networks [22]. Several new notions of weighted clustering coefficient have also been introduced ([2, 24] among others) but, unfortunately, no efficient method for estimating the weighted clustering coefficient has been presented so far.

Computing the exact values of the weighted clustering coefficient is at least as hard as for the unweighted clustering coefficient. Sampling is the key for an efficient and accurate approximation [7, 11]. In the unweighted case, the key to the design of an unbiased estimator is the ability of drawing uniformly at random a neighbor pair of a node and reporting 1 if and only if the neighbor pair is connected. The problem of drawing a neighbor pair can be efficiently solved in linear time if the two neighbors can be decided independently. The sampling of the two edges of a neighbor pair cannot be independent if the contribution to the clustering coefficient depends on the weights of the edges [2, 24] thus leading to a superlinear sampling complexity. Nevertheless in this paper we show that for several measures of weighted clustering coefficient it is possible to obtain an efficient linear time estimator.

Our Contributions We summarize in the following the main contributions of our work:

1. We show how to obtain efficient estimators for several standard definitions of weighted clustering coefficient.

2. Our sampling algorithm are easily parallelizable too. We also develop a scalable MapReduce implementation of our estimators Our implementation uses two rounds of MapReduce, it sends a number of messages across machines limited by the number of nodes times the number of samples required. The load for each machine is limited by the number of samples used by the algorithm times the maximum degree of a node in a graph.

3. We introduce a novel notion of *weighted clustering coefficient.* We base our proposal on the observation that edges with large weights are more likely to play a role in the social network. Our model defines a family of unweighted random graphs with edges existing with different probabilities. The probability of an edge depends on its weight. The largest the weight, the highest the probability. Each graph of the family of random graphs is an unweighted graph. The local *weighted clustering coefficient* of a node is defined as the expected local clustering coefficient in the family of random graphs. Our definition naturally extends to the weighted clustering coefficient of the entire graph.[1]

4. We also design a polynomial time algorithm to compute the value of the *weighted clustering coefficient* and a sampling technique to estimate it efficiently. The computation of the weighted clustering coefficient in our model does not require the generation of all graphs of the family. That would be computationally prohibitive. We show that a dynamic program is able to compute the exact local weighted clustering coefficient in polynomial time. [2] The computationally complexity of this exact computation is still prohibitive in practice. We are however able to design an efficient estimator also for this new definition of clustering coefficient.

5. We perform experiments that show interesting properties of the weighted clustering coefficient.

[1] We note that our definition of weighted random random graph is different from the definition of [10] and it is more in line with the standard definition used in data mining and biology [12].

[2] The problem of computing a core decomposition in an uncertain graph with different probabilities on edges has been considered in [5]. The authors show how to compute the expected degree of an uncertain graph in polynomial time. This method cannot however be applied to speed up the computation of the the novel notion of weighted clustering coefficient we propose.

1.1 Related works

A survey of several approaches to clustering coefficient in weighted networks can be found in [27]. In [25] the definition of clustering coefficient is based on the average weight on the edges of a triangle. In [2] the definition of the local clustering coefficient of a node only depends on the weights of the two edges incident to the node but not on the weight of the third edge of the triangle. In [21] it is adopted the standard unweighted definition with the exception that triangles are weighted by the edge that closes the triangle. In [24] the weight is only considered in the numerator of the definition of clustering coefficient whereas the denominator is the one of the unweighted case. In [37] the weight of a triangle is obtained by multiplying the weights of the edges. Other proposals that are substantially different from our approach can also be found in [15, 38]. The study of the clustering coefficient in several classes of random unweighted graphs can be found in [4].

The problem of estimating the clustering coefficient is closely related to the problem of counting the number of triangles in a graph. This is computationally expensive even on graphs of moderate size because of the time complexity needed to enumerate all the length-two paths of the graph. Several works proposed efficient heuristics [17, 29] with computational results reported for graphs of large size. More recently, there are algorithms designed under the MapReduce [9] programming model. Using a MapReduce infrastructure, [30] proposed algorithms for computing the exact number of triangles and the clustering coefficient of graphs. Randomized algorithms for counting triangles were also implemented under the MapReduce paradigm [26]. Finally to estimate the total number of triangle in a graph is possible to use also matrix sketches [20], unfortunately it is not clear how to extend this approach to local clustering coefficient. A related measure is also the transitivity coefficient of a graph [23]. Techniques adopted for estimating the clustering coefficient usually extend to the transitivity coefficient.

A natural approach for problems in massive networks is also to provide approximate solutions based on the application of data stream and random sampling algorithms. These algorithms usually provide an $(1 \pm \epsilon)$ approximation of the number of triangles with probability $1 - \delta$. The number of samples and amount of memory needed depends on the quality of the approximation. Data stream algorithms for estimating the number of triangles of a graph have been considered in [14, 33]. Semi-streaming algorithms have been proposed in [3]. A sampling-based algorithm for estimating the clustering coefficient of a graph is given in [28].

2 Preliminaries

Let $G = (V, E)$ be an undirected graph with $n = |V|$ and $m = |E|$ edges. For every vertex $v \in V$ let $\mathcal{N}(v, G)$ denote its neighborhood, i.e. $\mathcal{N}(v, G) =$

$\{u \in V : \exists(u,v) \in E\}$. The *clustering coefficient* $C_v(G)$ of a vertex $v \in V$ is defined as the probability that a random pair of its neighbors is connected by an edge, i.e. $C_v(G) := \frac{\left|\left\{(u,w)\in E: u,w\in\mathcal{N}(v,G)\right\}\right|}{\binom{|\mathcal{N}(v,G)|}{2}}$. In case of $|\mathcal{N}(v,G)| < 2$ we define $C_v(G) := 0$. The *clustering coefficient* $C(G)$ of G is the average clustering coefficient of its vertices, i.e. $C(G) = \frac{1}{n} \cdot \sum_{v\in V} C_v(G)$.

Let us denote by $W(v,G) = \{\langle u,w\rangle : u,w \in \mathcal{N}(v,G)\}$ the set of *wedges* of vertex v in graph G, i.e., the set of distinct paths of length two centered at v.

We denote by $w : E \to \Re^+$ the positive weight on the edges of the graph. Let $W = \max_{e\in E} w(e)$ be the maximum weight of an edge. We normalize the edge weights in a way that their range varies in $[0,1]$. Denote by $p : E \to [0,1]$ the normalized weights. We denote with $\mathbf{1}_C$, the indicator variable for the event C. In the experimental section we will use the following classic normalization $p(e) = \frac{1}{1+\log W/w(e)}$.

Finally, we say that we have an (ϵ,δ) estimator for a measure M, if we can estimate M within an ϵ multiplicative factor with probability at least $1-\delta$.

2.1 Generalizations of clustering coefficient in weighted networks

In this paper we consider three generalizations of the clustering coefficient in weighted networks. In particular we focus our attention to two definitions proposed in [2, 24] that well represent two general approaches to the problem: in one case the weights of the edges are added, in the other case they are multiplied. We additionally introduce a novel definition that is particularly relevant when the weights on the edges can be interpreted as probabilities[3].

Onnela et al. The first definition of clustering coefficient that we consider has been introduced by Onnela et al. [24]:

$$WC_v^{Onnela} = \frac{\sum_{\langle u,w\rangle\in W(v,G)} \hat{w}(e(v,u))\hat{w}(e(v,w))\hat{w}(e(u,w))}{|\mathcal{N}(v,G)|\left(|\mathcal{N}(v,G)|-1\right)}.$$

where with $w(e(v,u))$ we indicate the weight of the edge $e(v,u)$ and $\hat{w}(e(\cdot,\cdot)) = \frac{w(e(\cdot,\cdot))}{W}$.

Barrat et al. The second definition of clustering coefficient that we consider has been introduced by Barrat et al. [2]:

$$WC_v^{Barrat} = \frac{\sum_{\langle u,w\rangle\in W(v,G)}(w(e(v,u)) + w(e(v,w)))\mathbf{1}_{e(u,w)}}{\left(|\mathcal{N}(v,G)|-1\right)\left(\sum_{v\in e} w(e)\right)}.$$

[3] This setting is particularly relevant when graphs are generated using inference models [13].

where $\mathbf{1}_{e(u,w)}$ is equal 1 if the edge (u, w) exists and 0 otherwise.

Weighted clustering coefficient for probabilistic networks The last measure that we analyze is novel. The basic idea is that the normalized weights can be interpreted as probabilities of existence of the edges in the graph. More formally, define the class of random graph $\mathcal{G}_{n,p}$ with edge e appearing independently with probability $p(e)$. Each graph $G' = (V, E') \in \mathcal{G}_{n,p}$ is an edge subset E' of E. The probability of G' is $p(G') = \prod_{e \in E'} p(e) \prod_{e \notin E'} (1 - p(e))$.

The *weighted clustering coefficient* WC_v of a vertex $v \in V$ is defined as the expected clustering coefficient over the class of graphs $\mathcal{G}_{n,p}$: $WC_v^{random} = \mathrm{E}_{G' \in \mathcal{G}_{n,p}} C_v(G')$.

3 Computing the weighted clustering coefficient in probabilistic networks

In this section we give a polynomial algorithm to compute the new definition of weighted clustering coefficient efficiently. Note that at first sight our problem seems computationally very challenging because there are exponentially many possible realizations of the neighborhood of each node.[4]

Our first algorithmic contribution is to show that the problem is in P, we give an algorithm with complexity $O(|\mathcal{N}(v, G)|^4)$. Our algorithm is based on a dynamic program that computes incrementally the contribution of each neighbor pair to the clustering coefficient of each node.

Unfortunately our exact algorithm is too slow to run on real networks where the maximum degree is typically very large(in the order of millions for Twitter or Google+) fortunately in the next section we show that the new measure has an efficient (ϵ, δ) estimator.

Recall that the unweighted clustering coefficient of a node v is defined as the probability that a randomly selected pair of its neighbors is connected by an edge, based on this we can give an alternative definition of weighted clustering coefficient for probabilistic networks. Let $\chi(u, w)$ be a random variable that has value 1 if the randomly selected pair is (u, w) and 0 otherwise. We have: $C_v(G) := \sum_{u,w \in \mathcal{N}(v,G) \wedge (u,w) \in E} Pr(\chi(u, w) = 1)$. Where each pair is counted only once. In the following we shorten $\mathcal{N}(v, G')$ to $\mathcal{N}'(v)$. Using this definition we can rewrite the weighted clustering coefficient for v as: $WC_v^{random} = \mathrm{E}_{G' \in \mathcal{G}_{n,p}} \left[\sum_{u,w \in \mathcal{N}'(v) \wedge (u,w) \in E'} Pr(\chi(u, w) = 1 | G') \right]$.

Now by defining $\xi(u, w)$ as a random variable that has value 1 if and only if $u, w \in \mathcal{N}'(v) \wedge (u, w) \in E'$, and by denoting with $\mathbf{1}_{\xi(u,w)}$ its indicator

[4] Note that enumerating all the triangles in the graph would not work in this setting because of the dependency induced by the number of wedges in the realization of the random graph.

function, we have:

$$
\begin{aligned}
& WC_v^{random} \\
= \ & \mathrm{E}_{G' \in \mathcal{G}_{n,p}} \left[\sum_{u,w \in \mathcal{N}'(v) \wedge (u,w) \in E'} Pr(\chi(u,w) = 1 | G') \right] \\
= \ & \mathrm{E}_{G' \in \mathcal{G}_{n,p}} \left[\sum_{u,w \in \mathcal{N}(v)} \left(\mathbf{1}_{\xi(u,w)} Pr(\chi(u,w) = 1 | G') \right) \right] \\
= \ & \sum_{u,w \in \mathcal{N}(v)} \mathrm{E}_{G' \in \mathcal{G}_{n,p}} \left[\mathbf{1}_{\xi(u,w)} Pr(\chi(u,w) = 1 | G') \right] \\
= \ & \sum_{u,w \in \mathcal{N}(v)} \left(Pr(\xi(u,w) = 1) * \mathrm{E}_{G' \in \mathcal{G}_{n,p}} \left[\mathbf{1}_{\xi(u,w)} Pr(\chi(u,w) = 1 | G') \middle| \xi(u,w) = 1) \right] \right) \\
= \ & \sum_{u,w \in \mathcal{N}(v)} \left(Pr(u,w \in \mathcal{N}'(v) \wedge (u,w) \in E') * Pr(\chi(u,w) = 1 | \xi(u,w) = 1) \right)
\end{aligned}
$$

Now the first term of the sum can be easily computed because $Pr(u, w \in \mathcal{N}'(v) \wedge (u,w) \in E') = p(e_{u,v})p(e_{w,v})p(e_{w,u})$. The second term is still problematic. In fact $Pr(\chi(u,w) = 1|\xi(u,w) = 1)$ depends on all the possible instantiations of G' and so it potentially involve the computation of exponentially many terms.

In the following we show how to compute it efficiently using dynamic programming[5]. Note that $Pr(\chi(u,w) = 1|\xi(u,w) = 1) = Pr(\chi(u,w) = 1|u, w \in \mathcal{N}'(v))$ because the existence of the edge (u,w) does not change the probability of selecting u and w as random neighbors of v. And $Pr(\chi(u,w) = 1|u, w \in \mathcal{N}'(v))$ is the probability that a pair u, w of neighbors of v are selected conditioned on the fact that $u, w \in \mathcal{N}'(v)$.

To compute this probability we use the equivalence between the following two processes. The first one selects two elements uniformly at random without replacement from a set S, and the second one computes a random permutation of the elements in the set S and then returns the first two elements of the permutation.

Using this equivalence we can rephrase the probability $Pr(\chi(u,w) = 1|u, w \in \mathcal{N}'(v))$ as the probability that in a random permutation of the nodes in $\mathcal{N}(v)$, u and w are the two nodes with the smallest positions in $\mathcal{N}'(v)$. Note that for this to happen either u and w are the first two nodes in the permutation of the nodes in $\mathcal{N}(v)$, or all the nodes that are in positions smaller than u and w do not appear in $\mathcal{N}'(v)$.

[5] Unfortunately to the best of our knowledge, there is no analytic technique to estimate this quantity correctly or with a close approximation without using a dynamic programming.

Now leveraging on this fact, we give a quadratic dynamic program to compute $Pr(\chi(u,w) = 1 | \xi(u,w) = 1)$. Consider an arbitrary order to the nodes in $\mathcal{N}(v) \setminus \{u, w\}$, $z_1, z_2, ..., z_{|\mathcal{N}(v)|-2}$. In our algorithm we implicitly construct all the possible permutations incrementally and at the same time we estimate the probability that u, w are selected in each permutation. More specifically, initially we analyze the permutations containing only the elements $\{u, w\}$ then the ones containing the elements $\{u, w, z_1\}$, then the ones containing the elements $\{u, w, z_1, z_2\}$, and so on so for until we get the probability for each permutation containing all the elements in $\mathcal{N}(v)$.

The key ingredient of our algorithm is the following observation. Once we have computed the probability for all the permutations containing the nodes $\{u, w, z_1, z_2, ..., z_{i-1}\}$, to extend our computation to the permutations containing also the node z_i, we have to consider only two scenarios: in the first one z_i appears after u, w in the permutation in this case the probability that u and w are the nodes in $\mathcal{N}'(v)$ with the two smallest positions is the same. In the second one z_i appears before either of u or of w, conditioned on this event the probability that u and w are the nodes in $\mathcal{N}'(v)$ with the two smallest positions decreases by a multiplicative factor $1 - p(e_{v,z_i})$.

We are now ready to state our dynamic program more formally. Let M be a square matrix of dimension $|\mathcal{N}(v)| - 1$ such that $M_{i,j}$, for $j \leq i$, contains the probability that in a random permutation of nodes $\{u, w, z_1, z_2, ..., z_i\}$ u and w are preceded by exactly j elements in the permutation but they are in the first and second position when we consider the ordering induced only to nodes in $N'(v)$. Note that $M_{0,0}$ is equal to 1, because in this case we consider permutations containing only $\{u, w\}$. Similarly, we can compute $M_{1,0}$ and $M_{1,1}$. In particular, for $M_{1,0}$ we require that z_1 is in a position after u and w so we have $M_{1,0} = \frac{1}{3} M_{0,0}$. Instead, $M_{1,1} = \frac{2}{3}(1 - p(e_{v,z_i}))M_{0,0}$. More generally, we have that for $j \leq i$:

$$M_{i,j} = \begin{cases} \frac{i-1}{i+1} M_{i-1,0} & \text{if j = 0} \\ \frac{i-j-1}{i+1} M_{i-1,j} + \frac{j+1}{i+1} \overline{p}(e_{v,z_i}) M_{i-1,j-1} & \text{if j<i} \\ & \text{and j>0} \\ \frac{i}{i+1} \overline{p}(e_{v,z_i}) M_{i-1,j-1} & \text{if j = i} \end{cases}$$

Where $\overline{p}(*) = 1 - p(*)$.

Once we have computed the matrix M we can compute $Pr(\chi(u,w) = 1 | u, w \in \mathcal{N}'(v))$, in fact we have that: $Pr(\chi(u,w) = 1 | u, w \in \mathcal{N}'(v)) = \sum_{i=0}^{|\mathcal{N}'(v)|-2} M_{|\mathcal{N}'(v)|-2,i}$ So we have:

$$WC_v^{random} = \sum_{u,w \in \mathcal{N}(v)} \left(\frac{1}{2} p(e_{u,v}) p(e_{w,v}) p(e_{w,u}) * \left(\sum_{i=0}^{|\mathcal{N}'(v)|-2} M_{|\mathcal{N}'(v)|-2,i} \right) \right)$$

We summarize our algorithm to compute WC_v^{random} here:

Algorithm*(exact WC_v^{random})*

```
Input: The weighted subgraph induced by $v \cup \mathcal{N}(v)$.
Output: $WC_v^{random}$.
$WC_v^{random} = 0$.
for all $u, w \in \mathcal{N}(v)$ do
  Compute the matrix $M$ for $u, w$
  Using $M$, compute the probability $p$ that $(u, v, w)$ is a triangle and is selected
  $WC_v^{random} + = p$
Output $WC_v^{random}$.
```

Note that the above algorithm has complexity $O(|\mathcal{N}(v)|^4)$, so it is too slow to run on large networks. For this reason, we study in the next section efficient estimators of the weighted clustering coefficient.

4 Efficient Estimators for the Weighted Clustering Coefficient

We propose efficient (ϵ, δ) estimators for the various definition of weighted clustering coefficient. Our estimators that use basic concentration theory are similar to the one presented in [6, 32]. They are the first linear estimators for the weighted clustering coefficient to the best of our knowledge.

Onnela et al. Recall the definition of Onnela et al. [24] given in Section 2.

In this definition the weighted clustering coefficient is equal to the total normalized weight of the triangles containing v averaged by the number of wedges centered on v. Thus if we sample the wedges uniformly at random, using the Hoeffding bound and the fact that the normalized weights[6] are in $[0, 1]$, we get an efficient (ϵ, δ) estimator for WC_v^{Onnela}.

More formally, let $X_1, \ldots, X_s$ identical random variables that with probability $\frac{1}{|\mathcal{N}(v,G)|(|\mathcal{N}(v,G)|-1)}$ have value $\hat{w}(e(v,u))\ \hat{w}(e(v,w))\hat{w}(e(u,w))$ for every wedge $< u, w >$. Then, $E\left[\sum_{i=1}^{s} X_i\right] = sWC_v^{Onnela}$. Furthermore, by Hoeffding bound we have that: $P\left(\left|X - E\left[\sum_{i=1}^{s} X_i\right]\right| \leq \epsilon E\left[\sum_{i=1}^{s} X_i\right]\right) \leq e^{\frac{\epsilon^2 E[\sum_{i=1}^{s} X_i]}{3}} = e^{\frac{\epsilon s WC_v^{Onnela}}{3}}$ So if we want $\delta > e^{\frac{\epsilon s WC_v^{Onnela}}{3}}$, it suffices to have $s \in O(\log \frac{1}{\delta} \cdot \frac{1}{\epsilon^2 \cdot WC_v^{Onnela}})$ samples.

Lemma 4.1 *There is a sampling-based algorithm which with probability* $1 - \delta$ *returns a* $(1 \pm \epsilon)$*-approximation of the local weighted clustering coefficient*

[6] Note that for this to work it is fundamental that the weight on the edges have been normalized and so are in $[0, 1]$.

WC_v^{Onnela} of a vertex v of a weighted graph G. It needs $\mathcal{O}(\log \frac{1}{\delta} \cdot \frac{1}{\epsilon^2 \cdot WC_v^{Onnela}})$ samples.

In the following we present the pseudocode to compute this estimator:

```
Algorithm(sampling $WC_v^{Onnela}$ )
Input: The weighted subgraph induced by $v \cup \mathcal{N}(v)$.
Output: Approximate $WC_v^{Onnela}$.
for all $i = 1$ to $s$ do
  sample a random wedge $< u, w >$ uniformly from $\mathcal{N}(v)$
  If $(u, w) \in E$ then set $X_i \leftarrow w(u,v)^{1/3} w(u,w)^{1/3} w(v,w)^{1/3}$
  else set $X_i \leftarrow 0$
Output $X := \frac{1}{s} \cdot \sum_{i=1}^{s} X_i$.
```

Furthermore note that for the sampler we only need to be able to sample random wedges and this can be easily done in linear time.

Barrat et al. Recall the definition of Barrat et al. [2] given in Section 2. In this case the weighted clustering coefficient is not an explicit average so we cannot use the Hoeffding bound directly as before. Nevertheless note that we can define WC_v^{Barrat} as the average value of the random variable X where X has value $\mathbf{1}_{e(u,w)}$ with probability $\frac{w(e(v,u)) + w(e(v,w))}{(|\mathcal{N}(v,G)|-1)\left(\sum_{v \in e} w(e)\right)}$ for all $\langle u, w \rangle \in W(v, G)$.

Using this alternative definition combined with the Chernoff bound we get that by using k samples of the wedges weighted with the correct probability we can get good estimation of WC_v^{Barrat}.

More formally, let $X_1, \ldots, X_s$ identical random variable that with probability $\sum_{<u,w>:\mathbf{1}_{e(u,w)}=1} \frac{(w(e(v,u)) + w(e(v,w)))}{(|\mathcal{N}(v,G)|-1)\left(\sum_{v \in e} w(e)\right)}$ have value 1 or 0 otherwise. Then, $E\left[\sum_{i=1}^{s} X_i\right] = sWC_v^{Barrat}$. Furthermore by Chernoff bound we have that:

$$
\begin{aligned}
P\left(\left|X - E\left[\sum_{i=1}^{s} X_i\right]\right| \leq \epsilon E\left[\sum_{i=1}^{s} X_i\right]\right) &\leq e^{\frac{\epsilon^2 E\left[\sum_{i=1}^{s} X_i\right]}{3}} \\
&= e^{\frac{\epsilon s WC_v^{Barrat}}{3}}
\end{aligned}
$$

So if we want $\delta > e^{\frac{\epsilon s WC_v^{Barrat}}{3}}$, it suffuse to have $s \in O(\log \frac{1}{\delta} \cdot \frac{1}{\epsilon^2 \cdot WC_v^{Barrat}})$ samples.

Lemma 4.2 *There is a sampling-based algorithm which with probability $1-\delta$ returns a $(1 \pm \epsilon)$-approximation of the local weighted clustering coefficient WC_v^{Barrat} of a vertex v of a weighted graph G. It needs $\mathcal{O}(\log \frac{1}{\delta} \cdot \frac{1}{\epsilon^2 \cdot WC_v^{Barrat}})$ samples.*

At first sight it may look that we need quadratic time to sample a wedge with the correct probability, but also in this case it is possible to get a sample in linear time. In fact to sample an edge with the correct probability it is enough to sample the first edge e_1 with probability $p_1(e_1) = \frac{(|\mathcal{N}(v,G)|-2)w(e_1)+(\sum_{v\in e} w(e))}{2(|\mathcal{N}(v,G)|-1)(\sum_{v\in e} w(e))}$ and the second one e_2 with probability $p_2(e_2|e_1) = \frac{w(e_1)+w(e_2)}{(|\mathcal{N}(v,G)|-2)w(e_1)+(\sum_{v\in e} w(e))}$. The probability to sample pair e_1, e_2 is exactly $p_1(e_1)p_2(e_2|e_1) + p_1(e_2)p_2(e_1|e_2) = \frac{w(e_1)+w(e_2)}{(|\mathcal{N}(v,G)|-1)\sum_{v\in e} w(e)}$. On the other hand, it is also easy to verify that $\sum_e p_1(e) = 1$ and $\sum_{e\neq e_1} p_2(e|e_1) = 1$. We present the pseudocode also for this estimator:

```
Algorithm(sampling $WC_v^{Barrat}$ )
Input: The weighted subgraph induced by $v \cup \mathcal{N}(v)$.
Output: Approximate $WC_v^{Onnela}$.
for all $i = 1$ to $s$ do
    sample the first edge $e_1 = (u, v)$ of the wedge with probability equal to $p_1(e_1) = \frac{(|\mathcal{N}(v,G)|-2)w(e_1)+(\sum_{v\in e} w(e))}{2(|\mathcal{N}(v,G)|-1)(\sum_{v\in e} w(e))}$
    sample the second edge $e_2 = (u, w)$ of the wedge with probability equal to $p_2(e_2|e_1) = \frac{w(e_1)+w(e_2)}{(|\mathcal{N}(v,G)|-2)w(e_1)+(\sum_{v\in e} w(e))}$
    If $(u, w) \in E_i$ then set $X_i \leftarrow \frac{w(u,v)+w(v,w)}{(|\mathcal{N}(v,G)|-1)\sum_{v\in e} w(e)}$
    else set $X_i \leftarrow 0$
Output $X := \frac{1}{s} \cdot \sum_{i=1}^{s} X_i$.
```

Weighted clustering coefficient for probabilistic networks. The algorithm is based on sampling a random wedge $\langle u, w\rangle \in W(v, G')$ from a random graph $G' \in \mathcal{G}_{n,p}$ and checking whether $(u, w) \in G'$. The core idea of our sampler is to generate for a node v s neighbor realizations $\mathcal{N}(v)_1, \ldots, \mathcal{N}(v)_s$ uniformly at random from $\mathcal{G}_{n,p}$. Then for each realization sample a random wedge $< u, w >$ uniformly from $\mathcal{N}(v)_i$ and check if the wedge is part of a triangle in the realization. The estimation of the clustering coefficient is equal to the number of wedge that are part of a triangle divided by s.

```
Algorithm(sampling $WC_v^{random}$ )
Input: The weighted subgraph induced by $v \cup \mathcal{N}(v)$.
Output: Approximate $WC_v^{random}$.
Sample $s$ neighbor realization $\mathcal{N}(v)_1, \ldots, \mathcal{N}(v)_s$ uniformly at random from $\mathcal{G}_{n,p}$
for all $i = 1$ to $s$ do
    sample a random wedge $< u, w >$ uniformly from $\mathcal{N}(v)_i$
    If $(u, w) \in E_i$ then set $X_i \leftarrow 1$
    else set $X_i \leftarrow 0$
Output $X := \frac{1}{s} \cdot \sum_{i=1}^{s} X_i$.
```

For the sake of completeness we give a simple analysis of the algorithm below. We first show that the expected value of X_i is exactly WC_v^{random}.

We have for each $i \in \{1, \ldots, s\}$: $\mathrm{E}\left[X_i\right] = \mathrm{E}_{G' \in \mathcal{G}_{n,p}} \left[\frac{\left|\left\{(u,w) \in E' : u,w \in \mathcal{N}(v,G')\right\}\right|}{\binom{|\mathcal{N}(v,G')|}{2}} \right] = \mathrm{E}_{G' \in \mathcal{G}_{n,p}} \left[C_v(G')\right] = WC_v^{random}$.

Then we use the fact that for $0-1$ random variables we have $\mathbf{Var}\left[X_i\right] \leq \mathbf{E}[X_i^2] = \mathbf{E}[X_i] = WC_v^{random}$.

Now we analyze the variance of X. Since the X_i are mutually independent we get $\mathbf{Var}\left[X\right] = \mathbf{Var}\left[\frac{1}{s} \cdot \sum_{i=1}^{s} X_i\right] = \frac{1}{s^2} \cdot \sum_{i=1}^{s} \mathbf{Var}\left[X_i\right] \leq \frac{WC_v^{random}}{s}$. Finally, we can apply Chebyshev inequality. This gives us $\mathbf{Pr}\left[\left|X - \mathbf{E}[X]\right| \geq \epsilon \cdot \mathbf{E}[X]\right] \leq \frac{\mathbf{Var}[X]}{(\epsilon \cdot \mathbf{E}[X])^2} \leq \frac{WC_v^{random}}{s \cdot \epsilon^2 \cdot (WC_v^{random})^2} = \frac{1}{s \cdot \epsilon^2 \cdot WC_v^{random}}$.

If $s \geq \frac{3}{\epsilon^2 \cdot WC_v^{random}}$ then with probability $\frac{2}{3}$ the algorithm *sampling* WC_v^{random} approximates the weighted clustering coefficient of vertex v in G within a relative error of $(1 \pm \epsilon)$. In order to amplify the probability of success we run the algorithm $\Theta(\log \frac{1}{\delta})$ times and return the median of all results. This leads to the following corollary:

Lemma 4.3 *There is a sampling-based algorithm which with probability* $1-\delta$ *returns a* $(1 \pm \epsilon)$*-approximation on the local weighted clustering coefficient* WC_v^{random} *of a vertex* v *of a weighted graph* G*. It needs* $\mathcal{O}\left(\log \frac{1}{\delta} \cdot \frac{1}{\epsilon^2 \cdot WC_v^{random}}\right)$ *samples.*

5 Parallel implementation

In this section we first give a brief introduction to the MapReduce [9] framework and then we describe a highly optimized and scalable MapReduce implementation of our sampling algorithms. In particular, here we focus on parallelize the algorithm to estimate WC_v^{random}, the other sampling strategies can be parallelized in a very similar way. In the experimental section we report experimental results showing that this implementation is extremely fast in practice.

The MapReduce framework is designed to simplify the implementation of parallel algorithms at very large scale. In the MapReduce framework the data is processed in tuples composed by $\langle key, value \rangle$. The computation proceeds in rounds. In the Map phase, each machine receives all the values associated with a specific *key* k, then it executes some computation and output $\langle key, value \rangle$ tuples with potentially different *key* k'. A Shuffle phase aggregates all tuples with same *key* k' that are sent to the same physical machine. Finally, in the the Reduce phase, each machine performs a computation that only depends from the tuples with same *key* k' outputted from the Mapper, and output $\langle key, value \rangle$ tuples with *key* equal to the input k'.

To write a MapReduce program it is typically important to design an algorithm that: i) minimizes the number of MapReduce rounds that are involved; ii) minimizes the amount of communication between machines; and iii) balances the working load across different machines. In the following we show how these requirements are achieved in the implementation of our sampling algorithm in MapReduce.

We assume that the input graph is stored in $\langle key, value\rangle$ tuples, representing the adjacency list of each node. In the first Map phase each machine reads the adjacency list of a node u. For sample $i = 1, \ldots, s$, the machine constructs a realization of the neighborhood of a node u, $\mathcal{N}_i(u)$, according to $\mathcal{G}_{n,p}$ and samples a pair of random neighbors $(v_i, w_i) \in \mathcal{N}_i(u)$. Then it sends a message with key w_i and value $i, (u, v_i)$ to the machine that controls node w_i [7] . The informal meaning of these messages is that node u asks node w_i whether edge (w_i, v_i) exists in the i-th realization so that we can infer that triangle u, v_i, w_i exists in realization i. Finally, node u sends also its adjacency list to itself in order to be able to answer the requests of other nodes.

In the first Reduce phase node u receives its own adjacency list and various requests $i, (w, v_i)$ to check the existence of edge (u, v_i) in realization $\mathcal{N}_i(u)$. If the test is positive, it writes a value $\langle u, w\rangle$ with its own key indicating that it is incident to a triangle with node w in one of the samples.

In the second Map phase each node v reads the values written in the previous Reduce phase and for each detected triangle $\langle v, u\rangle$ sends a message $\langle u, 1\rangle$ to the other node u certifying the existence of the triangle. Finally, in the last Reduce step each node receives the number of sampled triangles and simply computes its clustering coefficient by dividing it by the number of samples.

The implementation presented above uses two rounds of MapReduce, it sends a number of messages across machines upper bounded by the number of nodes times the number of samples required. The load for each machine is upper bounded by the number of samples used by the algorithm times the maximum degree of a node in a graph.

6 Experiments

The main goal of this section is to show experimentally some properties of the weighted clustering coefficient and to show the speed-up obtained by our simple estimators.

[7] Note that a naive implementation of the sampling procedure would have running time quadratic in the size of the adjacency list. Fortunately this is not necessary in fact it is possible to select a random pair of neighbors in linear time. In particular, it is enough to assign to each neighbor a random number and then select the two neighbors with the smallest assigned values. In this way each pair of nodes has the same probability of being selected and so we can obtain a random sample.

We start by describing a classical application of the clustering coefficient and the dataset that we will use in our experiment.

Then we analyze experimentally different techniques to map integer weights to weights between $[0, 1]$. In particular we compare two mapping techniques and we analyze the trade-off between them. Note that this mapping is needed for two of our three clustering coefficient measures[8].

We then compare the performance of the three weighted clustering coefficient measures with the classic unweighted clustering. Our findings are quite encouraging, in fact we observe that the weighted clustering performance is always at least as good as the unweighted clustering coefficient and our new notion of weighted clustering coefficient "outperforms" the classic unweighted notion.

Finally we analyze the scalability of our approach. In particular we run our algorithm using different number of machines on networks of increasing sizes. We observe that our algorithm is highly scalable and it can fully leverage on parallelization to improve its performances.

Dataset and experiment settings. The clustering coefficient is a fundamental topological property of networks and also one of the most used topological features in machine learning on graphs. Indeed, it has been used to detect spam on the web [3] and malicious users in social networks [36].

For this reason we study the effectiveness of weighted clustering coefficient by studying its power as a machine learning feature. In particular, we focus on the specific case where we are interested in detecting spam in the Web. Toward this end, we use a public available dataset [8] composed by a collection of hosts manually labelled (spam/non spam) by a group of volunteers and by the weighted host graph network. The graph is composed by 114,529 hosts in the .UK domain and there are 5709 hosts marked as "non spam" and 344 hosts marked as "spam". Even if the web graph is directed in this section we ignore the directionality of the edges for simplicity[9]. Finally we note that there are 2058 hosts marked as "non spam" and 93 hosts marked as "spam" with clustering coefficient bigger than 0 (for any, weighted or unweighted, definition of clustering coefficient).

In our experiments we are only interested in analyze the correlation between various definitions of clustering coefficient and the integrity of an host. To do it, for each definition we first compute the corresponding score for each labelled node, then we rank all the labelled nodes with score bigger than 0 according to their scores and compute the precision of each position i of the ranking as the percentage of "non spam" hosts before position i. This

[8] Experimentally we observed that a nonlinear mapping perform better than the linear mapping proposed by Onnela et al.

[9] Note that all the discussed notion of clustering coefficient can be extended to capture the directionality of the edges.

measure, even if simplistic, gives a good intuition of the correlation between the clustering coefficient and the goodness of a page.

Finally to analyze the scalability of our algorithm we consider five graphs from the SNAP repository: Friendster, Orkut, LiveJournal, Pokec and Patents. Those graphs are unweighted so we assign a random weight between $[0, 1]$ to every edge independently. In Table **??** we report some basic statistic on the graphs.

Dataset	Nodes	Edges
Patents[19]	3,774,768	16,518,948
Pokec[31]	1,632,803	30,622,564
LiveJournal[1]	4,847,571	68,993,773
Orkut[35]	3,072,441	117,185,083
Friendster[35]	65,608,366	1,806,067,135

Table 1: Network statistics.

6.1 Building the probabilistic graph

Two of the analyzed definitions of weighted clustering coefficient cannot be applied if the edges' weights are not in $[0, 1]$. In this subsection we analyze different techniques to build a probabilistic graph from the input graph by mapping the weights to probability in $[0, 1]$. As a case of study we analyze the effect of different mappings on the .UK domain graph.

Let us define e_W and e_w, respectively the maximum and the minimum weight of an edge in the input graph. A first natural technique to construct our probabilistic graph is to use a linear mapping between $[e_w, e_W]$ and $[0, 1]$. This mapping although has a serious drawback for our probabilistic definition: in practice the weights on the edges are distributed as a power law and so e_w/e_W is very small (for example in our case of study is $1/2579857$). So if we use this mapping we would end up to have very small weights on the edges of the graph and this this would in turn imply an extremely small realization probability for every triangle in the graph.

To solve this issue in our experiment we consider two non-linear mapping functions M_1, M_2. Both functions are mapping between $[e_w, e_W]$ and $[0, 1]$, more formally we have that both $M_1, M_2 : [e_w, e_W] \to [0, 1]$. In particular we define $M_1(w) = \frac{\log(w - e_w + 1)}{\log(e_W - e_w + 1)}$ and $M_2(w) = \frac{1}{1 + \log\left(\frac{e_W - e_w + 1}{w - e_w + 1}\right)}$. To compare those two mappings we run our approximation algorithm for estimating WC_v^{random} for all the nodes in the graph and we compare the precision of rankings that we obtain by using the two different rankings. In this experiment to compute the weighted clustering coefficient we execute 3200 samples per node and to compute the average precision and the standard deviation we rerun the algorithm 4 times with different random seeds. In Figure 1 and in the rest of this section we plot the average precisions using lines and the standard deviations using shadows around the lines.

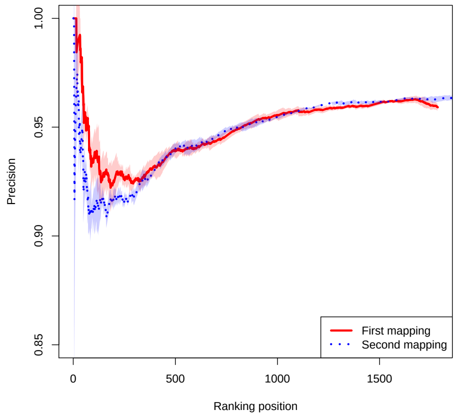

Figure 1: Comparison between the two mapping strategies M_1 and M_2.

From the experiments it is possible to conclude that the two mapping strategies have similar performances although M_1 seems to perform slightly better. For this reason for the rest of this section we then focus only on results obtained using the mapping M_1.

6.2 Performances of the sampling algorithm.

Now that we have defined our mapping strategy and built our probabilistic graph we can focus on the performance of our sampling algorithm. Here we first analyze the running time of the sampling algorithm presented in Section 4 when we vary the number of samples used in the algorithm and we compare its running time with the running time of the algorithms that consider all the triangles to compute the unweighted clustering coefficient or the weighted clustering coefficient defined by Barrat et al. [2].

Then we analyze how the precision of the ranking varies as a function of the number of samples performed by the algorithm. In Figure 2 we show the average running time of the sampling algorithm when we vary the number of samples relative to the running time of the optimal algorithm. It is interesting to note that the running time increase almost linearly with the number of seeds showing that the algorithm efficiently use all the parallelization offered by the MapReduce framework. Furthermore it is quite interesting to note the huge difference in running time between the sampling algorithm and the quadratic algorithm that considers all the triangles. In fact when we used $50, 100, 200$ and 400 samples the sampling algorithm is

900 times faster than the quadratic algorithm, and even when we use 2000 samples the sampling algorithm is still 120 times faster!

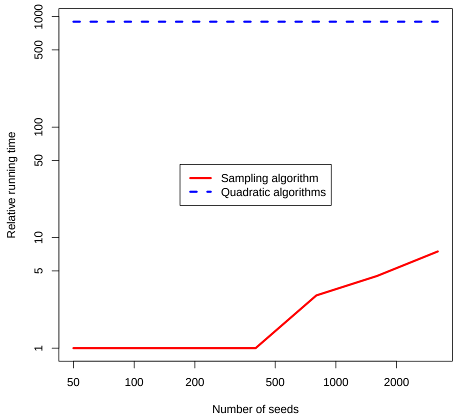

Figure 2: Running time *vs* Number of seeds and precision when we vary the number of samples between 50, 100, 200 and 3200.

Now we turn our attention to the effects of varying the number of samples on the precision of the algorithm. In Figure 3 we show how the precision curve of the new notion of weighted clustering coefficient changes when we use 50, 100, 200 or 3200 samples(we notice a similar trends also with 400, 800, 1600 samples and for other clustering coefficient definition, we do not show them in the figure for clarity). Also in this case we plot the average precisions with lines and the standard deviations with the shadows around the lines. From the plots it seems that few samples are enough to obtain a good estimation of the weighted clustering coefficient.

There are several interesting observations to make here.

First, as predicted from Theorem 4.3, for all the measures the standard deviation decreases very quickly as the number of samples used by our algorithm increase.

Second, for WC_v^{random} the length of the ranking computed by our algorithm decrease when we use a smaller number of samples. This is probably due to the fact that several nodes are incident to a small number of triangles and so by executing a small number of samples we do not discover them.

The third observation is probably the most striking: the precision of ranking provided by WC_v^{random} decreases when we use larger numbers of samples. We hypothesize that this phenomena can be explained using the same explanation that we used for our second observation. In fact, also in this case nodes that have small degrees are not likely to appear in the ranking when we consider few samples. But for nodes of small degrees the clustering coefficient is probably not a meaningful indicator of their trustfulness. To

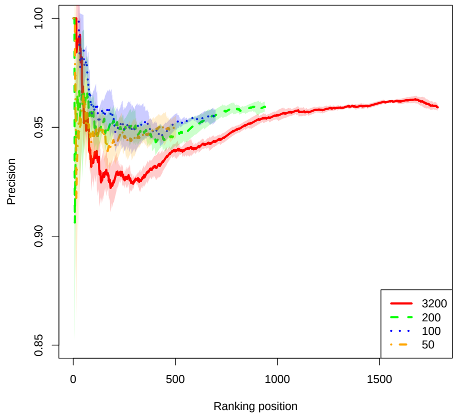

Figure 3: Precision *vs* Ranking position of the sample algorithm when we vary the number of samples between 50, 100, 200 and 3200.

verify this hypothesis in the next subsection we analyze how the precision of the rankings changes when we consider only nodes with degrees above a specified threshold.

Motivated from last observation, here we analyze the relationship between the degree of a node and the correlation between its clustering coefficient and its trustfulness for WC_v^{random}. To do this, we analyze the precision of the rankings of nodes when we restrict only to nodes with weighted or unweighted degree above a specific threshold.

In Figure 4 we analyze the precision of the rankings computed by our sampling algorithm by using 3200 samples when we restrict to nodes with unweighted degree at least $0, 5, 10$ and 20.

We observe a trend similar to the one observed in Figure 3, suggesting that there is an interesting relationship between the degree of a node and the correlation of its weighted clustering coefficient with its trustfulness.

6.3 Comparison between different definitions

In this subsection we compare the three definitions of weighted clustering coefficient with the classic definition of unweighted clustering coefficient. We show that the new definition is always comparable with the other two and in various point of the ranking it perform significantly better.

In Figure 5 we show the ranking obtained using the four definitions. For the classic unweighted definition we compute the clustering coefficient of each node exactly. For the three weighted clustering coefficient definition we approximate the clustering coefficient using 3200 samples per nodes.

It is possible to note that the two previous definitions of clustering co-

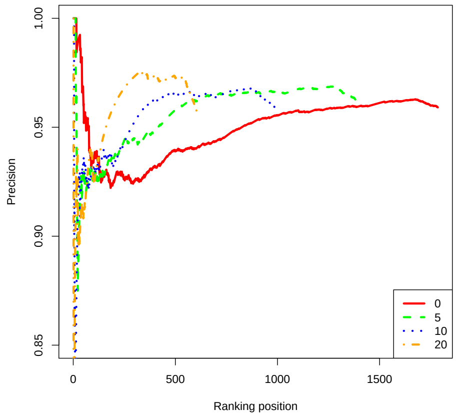

Figure 4: Precision of the sample algorithm when we restrict only to nodes with unweighted degree bigger than $0, 5, 10$ and 20 for WC_v^{random}.

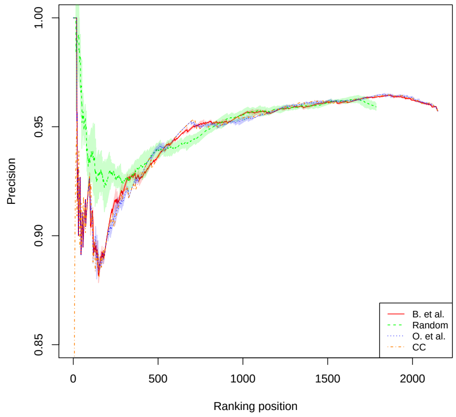

Figure 5: Comparison between the precision rankings obtained by classic definition of clustering coefficient(CC), the definition by Barrat et al.(B. et al), the definition of Onnela et al. (O. et al), and our new definition(Random).

efficient have very similar performances and performances very similar to the classic unweighted definition while the ranking obtained by our new definition has higher precision for the first positions in the ranking and then has performances comparable with the rankings obtained using the other definitions.

Finally, we compare the performances of our new definition with the performances of the definition given by Barrat et al. when we restrict to nodes with unweighted degree above a specific threshold. We think that this case is of particular interest because we showed in the previous subsection there is an interesting relationship between the degree of a node and the correlation between its weighted clustering coefficient and its trustfulness.

In Figure 6 we present the comparison between the two definition when we restrict our attention to nodes of degree larger than $0, 5$ and 20.

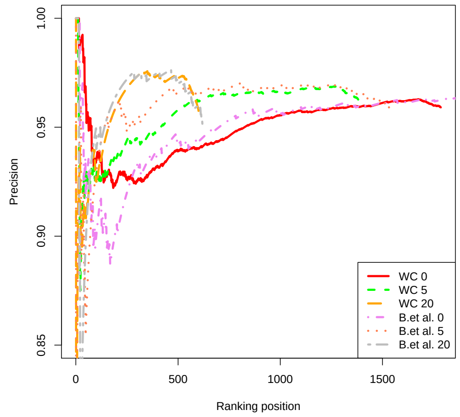

Figure 6: Comparison between the precision rankings obtained by the definition by Barrat et al.(B. et al) and our new definition(WC) when we restrict to node of degree larger than $0, 5$ and 20.

Also in this case we observe that the two definitions have very similar performances.

6.4 Scalability of our algorithm

In this final subsection we analyze the scalability of our new algorithm. In order to do it, we analyze consider five public datasets available in the SNAP repository: Orkut, Patents, Pokec, LiveJournal and Friendster. The five graphs have an increasing number of edges(each dataset has roughly twice as many edges as the previous one). The input dataset are unweighted, so before run our algorithm we assign a random weight between 0 and 1 to every edge.

In this experiment we are interested in analyzing the running time of the algorithm when we increase the number of machines available and when we increase the size of the network analyzed.

In Figure 7 we present the running time of the algorithm on different networks with different resources. Note that all the number show in the plots are relative. In particular on the y axis we present the running times as the relative running time in comparison with the fastest run of algorithm on the smallest graph and on the x axis we present the number as multiplicative factor of the minimum number of machines used.

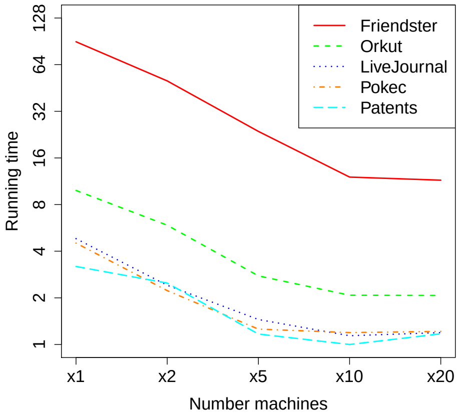

Figure 7: Running time of the algorithm on different networks with different resources. Note that all the number show in the plots are relative. In particular on the y axis we present the running times as the relative running time in comparison with the fastest run of algorithm on the smallest graph and on the x axis we present the number as multiplicative factor of the minimum number of machines used $x2, x5, x10$ and $x20$

It is interesting to note that our algorithm is able to leverage on parallelization to speed-up computation on very large graphs. Note, for example, that by increasing the number of machines by a factor of 10 it is possible to reduce the running time on the Friendster graph of roughly a factor of 10. It is also worth noticing that we do not obtain much gains by increasing the number of machines by a factor of 20, this is probably due to a trade-off between computational power and cost of communication within machines.

7 Conclusions

In this work we present sampling techniques to obtain efficient estimators for several measures of weighted clustering coefficient together with their Map Rreduce implementation. We also propose a novel graph-theoretic notion of clustering coefficient in weighted networks defined as the expected unweighted clustering coefficient on a family of random graphs. Moreover, we show on an application related to web spam detection that the notions of weighted clustering coefficient compare with the standard notion of unweighted clustering coefficient as a machine learning feature to assess the quality of nodes in a social network. Given the importane of weighted networks to model the strength of the interaction between nodes in a graph, we hope that our work will prompt more study on the relevance of weighted graph mining features to characterize the inner structure of social networks.

Acknowledgments We thank Corinna Cortes for suggesting the problem.

References

[1] L. Backstrom, D. Huttenlocher, J. Kleinberg, and X. Lan. Group formation in large social networks: Membership, growth, and evolution. In *KDD*, 2006.

[2] A. Barrat, M. Barthlemy, R. Pastor-Satorras, and A. Vespignani. The architecture of complex weighted networks. *Proceedings of the National Academy of Sciences of the United States of America.*

[3] L. Becchetti, P. Boldi, C. Castillo, and A. Gionis. Efficient semi-streaming algorithms for local triangle counting in massive graphs. In *KDD'08.*

[4] B. Bollobas. Mathematical results on scale-free random graphs. In *Handbook of Graphs and Networks.*

[5] F. Bonchi, F. Gullo, A. Kaltenbrunner, and Y. Volkovich. Core decomposition of uncertain graphs. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 1316–1325, New York, NY, USA, 2014. ACM.

[6] C. Budak, D. Agrawal, and A. El Abbadi. Structural trend analysis for online social networks. *VLDB'11.*

[7] L. Buriol, G. Frahling, S. Leonardi, A. Marchetti-Spaccamela, and C. Sohler. Counting triangles in data streams. In *PODS 06.*

[8] C. Castillo, D. Donato, L. Becchetti, P. Boldi, S. Leonardi, M. Santini, and S. Vigna. A reference collection for web spam. *SIGIR '06.*

[9] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI '04.*

[10] G. Fagiolo. Clustering in complex directed networks. *Phys. Rev. E.*

[11] S. J. Hardiman and L. Katzir. Estimating clustering coefficients and size of social networks via random walk. In *WWW '13.*

[12] P. Hintsanen and H. Toivonen. Finding reliable subgraphs from large probabilistic graphs. *Data Min. Knowl. Discov.*

[13] P. Hintsanen and H. Toivonen. Finding reliable subgraphs from large probabilistic graphs. *Data Min. Knowl. Discov.*, 2008.

[14] M. Jha, C. Seshadhri, and A. Pinar. A space efficient streaming algorithm for triangle counting using the birthday paradox. In *KDD '13.*

[15] G. Kalna and D. J. Higham. Clustering coefficients for weighted networks. In *Symposium on Network Analysis in Natural Sciences and Engineering.*

[16] H. Kwak, C. Lee, H. Park, and S. Moon. What is twitter, a social network or a news media? In *WWW '10.*

[17] M. Latapy. Main-memory triangle computations for very large (sparse(power-law)) graphs. *Theoretical Computer Science.*

[18] J. Leskovec and E. Horvitz. Planetary-scale views on a large instant-messaging network.

[19] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graphs over time: Densification laws, shrinking diameters and possible explanations. In *KDD*, 2005.

[20] E. Liberty. Simple and deterministic matrix sketches. In *KDD*, 2014.

[21] L. Lopez-Fernandez, G. Robles, and J. M. Gonzalez-Barahona. Applying social network analysis to the information in cvs repositories. In *1st International Workshop on Mining Software Repositories (MSR).*

[22] M. E. J. Newman. Analysis of weighted networks. *Phys. Rev. E*, 70:056131, Nov 2004.

[23] M. E. J. Newman, D. J. Watts, and S. H. Strogatz. Random graph models of social networks. *Proc. Natl. Acad. Sci. USA*, 99:2566–2572, 2002.

[24] J.-P. Onnela, J. Saramäki, J. Kertész, and K. Kaski. Intensity and coherence of motifs in weighted complex networks. *Physical Review E.*

[25] T. Opsahl and P. Panzarasa. Clustering in weighted networks. *Social Networks.*

[26] R. Pagh and C. E. Tsourakakis. Colorful triangle counting and a mapreduce implementation.

[27] J. Saramäki, M. Kivelä, J.-P. Onnela, K. Kaski, and J. Kertesz. Generalizations of the clustering coefficient to weighted complex networks. *Physical Review E.*

[28] T. Schank and D. Wagner. Approximating clustering coefficient and transitivity. *Journal of Graph Algorithms and Applications.*

[29] T. Schank and D. Wagner. Finding, counting and listing all triangles in large graphs, an experimental study. In *WEA '05.*

[30] S. Suri and S. Vassilvitskii. Counting triangles and the curse of the last reducer. In *WWW' 11.*

[31] L. Takac and M. Zabovsky. Data analysis in public social networks. In *International Scientific Conference & International Workshop Present Day Trends of Innovations*, 2012.

[32] C. E. Tsourakakis, U. Kang, G. L. Miller, and C. Faloutsos. Doulion: counting triangles in massive graphs with a coin. In *KDD '09.*

[33] C. E. Tsourakakis, M. N. Kolountzakis, and G. L. Miller. Triangle sparsifiers. *J. Graph Algorithms Appl.*

[34] D. J. Watts and S. H. Strogatz. Collective dynamics of small-world networks. *Nature.*

[35] J. Yang and J. Leskovec. Defining and evaluating network communities based on ground-truth. In *ICDM*, 2012.

[36] Z. Yang, C. Wilson, X. Wang, T. Gao, B. Y. Zhao, and Y. Dai. Uncovering social network sybils in the wild. In *IMF '11.*

[37] B. Zhang, S. Horvath, et al. A general framework for weighted gene co-expression network analysis. *Statistical applications in genetics and molecular biology.*

[38] Y. Zhang, Z. Zhang, J. Guan, and S. Zhou. Analytic solution to clustering coefficients on weighted networks. *arXiv preprint arXiv:0911.0476*, 2009.