Tracking Clustering Coefficient on Dynamic Graph via Incremental Random Walk

Qun Liao, Lei Sun, Yunpeng Yuan, and Yulu Yang(✉)

College of Computer and Control Engineering, Nankai University, Tianjin, China
{liaoqun, sunleier, yuanyp}@mail.nankai.edu.cn, yangyl@nankai.edu.cn

Abstract. Clustering coefficient is an important measure in complex graph analysis. Tracking clustering coefficient on dynamic graphs, such as Web, social networks and mobile networks, can help in spam detection, community mining and many other applications. However, it is expensive to compute clustering coefficient for real-world graphs, especially for large and evolving graphs. Aiming to track the clustering coefficient on dynamic graph efficiently, we propose an incremental algorithm. It estimates the average and global clustering coefficient via random walk and stores the random walk path. As the graph evolves, the proposed algorithm reconstructs the stored random walk path and updates the estimates incrementally. Theoretical analysis indicates that the proposed algorithm is practical and efficient. Extensive experiments on real-world graphs also demonstrate that the proposed algorithm performs as well as a state-of-art random walk based algorithm in accuracy and reduces the running time of tracking the clustering coefficient on evolving graphs significantly.

Keywords: Clustering coefficient · Graph mining · Incremental algorithm · Random walk

1 Introduction

In the last decades, clustering coefficient [1] has emerged as an important measure of the homophily and transitivity of a network. Tracking clustering coefficient for networks which are large and evolving rapidly, such as World Wide Web and online social networks, is an important issue for many real-world applications, such as detecting spammer for search engines [2] or online video networks [3] and detecting events in phone networks [4].

However, there are two challenges in tracking clustering coefficient. First, computing clustering coefficient for evolving graphs with millions or billions of edges is time-consuming. Especially, the efficient algorithms for static graphs [5, 6] are not sufficient for evolving graphs. Second, for most real-world networks, such as online social networks, the network topology and the size of network are evolving and can not be known beforehand. Moreover, it is always limited restrictedly to access the underlying network for the sake of performance or safety. Thus, the basic assumption of "independent random sampling" in most random sampling based algorithms [7, 8] is not realistic.

© Springer International Publishing AG 2017
A. Bouguettaya et al. (Eds.): WISE 2017, Part I, LNCS 10569, pp. 488–496, 2017.
DOI: 10.1007/978-3-319-68783-4_33

Aiming to track clustering coefficient on large and evolving networks, we propose an incremental algorithm. The proposed algorithm follows a random walk based sampling method to estimate clustering coefficient [9], only relying on the public interface and requiring no prior knowledge. The proposed algorithm reuses previous random walk and gets result updated via reconstructing partial random walk as the graph evolves, instead of recomputing from scratch. Both theoretical analysis and experimental evaluations on real-world graphs demonstrate that the proposed algorithm reduces the running time significantly comparing with a state-of-art random walk sampling based algorithm without sacrificing accuracy.

2 Tracking Clustering Coefficient

2.1 Clustering Coefficient

Let $G = (V, E)$ stand for a simple graph with n vertices and m undirected edges. The ith vertex is donated by v_i, $1 \leq i \leq n$. The edge between v_i and v_j is donated by e_{ij}, or e_{ji}. The degree of vertex v_i is donated by d_i. The adjacency matrix of G, donated by A, is a $n \times n$ symmetric matrix, $A_{i,j} = A_{j,i} = 1$ if and only if there is an edge between v_i and v_j, and $A_{i,j} = A_{j,i} = 0$ otherwise. We assume that there are no edges from any arbitrary vertex v_i pointing to itself, thus $A_{i,i} = 0$, $1 \leq i \leq n$.

We define a wedge as a triplet (v_j, v_i, v_k) for any i, j and $k \in [1, n]$, if and only if $A_{i,j} = A_{i,k} = 1$, and $j < k$. For any wedge (v_j, v_i, v_k) with $A_{j,k} = 1$, we define it a triangle. Let l_i stand for the number of triangles with v_i lying in the middle. It is obvious that l_i is equal to the number of connected edges between v_i's neighbors.

The local clustering coefficient for any node v_i, donated by c_i, is the ratio of the number of edges between v_i's neighbors to the maximal possible number of such edges. For node v_i where $d_i \geq 2$, we define c_i as $2l_i/d_i(d_i - 1)$. For node v_i with less than 2 neighbors, c_i is defined as 0.

In this paper, we focus on two popular versions of clustering coefficient: the average clustering coefficient [10] and the global clustering coefficient [10]. The average clustering coefficient of a graph, donated by a, is the average of the local clustering coefficient over the set of nodes. It is defined as $\sum_{i=1}^{n} c_i/n$. The global clustering coefficient, also called transitivity in some previous works, is a metric measuring the probability that two neighbors of a node are connected to one another in global. In this paper, we donate it by g and define it as $2\sum_{i=1}^{n} l_i / \sum_{i=1}^{n} d_i(d_i - 1)$.

2.2 Problem Definition

Let $G(t)$ stand for a snapshot of an evolving graph at time t, $0 \leq t$. $G(0)$ is the initial graph. For $t > 0$, $G(t)$ is a graph with an arbitrary edge inserted into (or removed from) $G(t-1)$. The average and global clustering coefficient of $G(t)$ are donated by $a(t)$ and $g(t)$ respectively. Their estimates are donated by $\widehat{a}(t)$ and $\widehat{g}(t)$ respectively. Our goal is to compute $\widehat{a}(t)$ and $\widehat{g}(t)$ efficiently, $0 < t$.

It is supposed that $\widehat{a}(0)$ and $\widehat{g}(0)$ are computed via the algorithm in [9] (hereinafter referred to as baseline algorithm). It generates a random walk with r steps, donated by $R(0) = \{x_1(0), x_2(0),\ldots, x_r(0)\}$. A variable $\varphi_k(t)$ is defined as $A_{x_{k-1}(t),x_{k+1}(t)}$, $2 \leq k \leq r-1, 0 \leq t$. There are another four variables defined as follows. $\Phi_a(0)$ is defined as $(r-2)^{-1}\sum_{k=2}^{r-1} \phi_k(0)\left(d_{x_k(0)} - 1\right)^{-1}$, $\Psi_a(0)$ is defined as $r^{-1}\sum_{k=1}^{r}\left(d_{x_k(0)}\right)^{-1}$, $\Phi_g(0)$ is defined as $(r-2)^{-1}\sum_{k=2}^{r-1} \phi_k(0)d_{x_k(0)}$ and $\Psi_g(0)$ is defined as $r^{-1}\sum_{k=1}^{r}\left(d_{x_k(0)} - 1\right)$. The approximate average and global clustering coefficient for $G(0)$, are defined as $\Phi_a(0)/\Psi_a(0)$ and $\Phi_g(0)/\Psi_g(0)$ respectively. It is assumed that for any t ($0 < t$), $R(t-1)$, $\Phi_a(t-1)$, $\Psi_a(t-1)$, $\Phi_g(t-1)$ and $\Psi_g(t-1)$ are stored.

2.3 Tracking Clustering Coefficient Incrementally

Without loss of generality, we present how the proposed algorithm works at time t ($0 < t$). It is supposed that e_{uv} is added at time t. The proposed algorithm checks whether v_u and v_v accessed by the stored path $R(t-1)$. If neither v_u nor v_v is accessed, it returns the previous results $\widehat{a}(t-1)$ and $\widehat{g}(t-1)$ without any updating. Otherwise, the proposed algorithm finds set of steps in $R(t-1)$ where either v_u or v_v accessed, donated by $X(t-1)$. $X(t-1) = \{x_k(t-1) \mid x_k(t-1) = v_u \,\|\, x_k(t-1) = v_v\}$. For each entry $x_k(t-1)$ in $X(t-1)$, it takes a chance to reroute the random walk and update the estimates. The probability of rerouting the random walk is $1/d_{x_k(t-1)}$.

As rerouting the random walk $R(t-1)$ from the kth step, there are two phases, as shown in Fig. 1. Firstly, the proposed algorithm undoes the random walk from $x_k(t-1)$, removing $r-k$ steps in $R(t-1)$ from $x_k(t-1)$. Secondly, the proposed algorithm regenerates random walk with $(r-k)$ steps starting form the $(k+1)$th step and gets the estimates updated. As v_u (or v_v) is accessed at the kth step, the proposed algorithm picks v_v (or v_u) at the regenerated $(k+1)$th step, then moves to one of its neighbors uniformly at random and repeats such random movement for $(r-k-1)$ times.

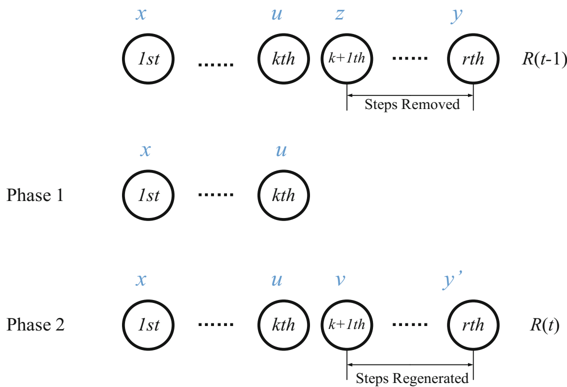


Fig. 1. Two phases in rerouting the random walk

We donate the set of removed steps by $R^{-} = \{x_h(t\text{-}1)\}$ and the added steps by $R^{+} = \{x_h(t)\}$, where $k < h \leq r$. $R(t) = R(t-1) - R^{-} + R^{+}$. And the variables are updated as follows.

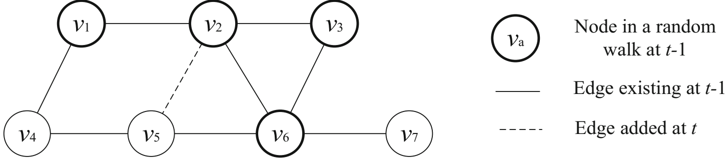
$$\Phi_a(t) = \Phi_a(t-1) - (r-2)^{-1}\sum_{x_k \in R^-} \phi_k(t-1)(d_{x_k}-1)^{-1} + (r-2)^{-1}\sum_{x_k \in R^+} \phi_k(t)(d_{x_k}-1)^{-1} \quad (1)$$

$$\Psi_a(t) = \Psi_a(t-1) - r^{-1}\sum_{x_k \in R^-} (d_{x_k})^{-1} + r^{-1}\sum_{x_k \in R^+} (d_{x_k})^{-1} \quad (2)$$

$$\Phi_g(t) = \Phi_g(t-1) - (r-2)^{-1}\sum_{x_k \in R^-} \phi_k(t-1)d_{x_k} + (r-2)^{-1}\sum_{x_k \in R^+} \phi_k(t)d_{x_k} \quad (3)$$

$$\Psi_g(t) = \Psi_g(t-1) - r^{-1}\sum_{x_k \in R^-} (d_{x_k}-1) + r^{-1}\sum_{x_k \in R^+} (d_{x_k}-1) \quad (4)$$

To make how the proposed algorithm works clear, we provide an example. As depicted in Fig. 2, $G(t) = G(t-1) + \{e_{25}\}$ and $R(t-1) = \{v_3, v_6, v_2, v_1\}$. The proposed algorithm reroutes $R(t-1)$ at the third step (where v_2 accessed) with 1/3 probability. Supposing we reroutes $R(t-1)$, we remove $R^{-}=\{v_1\}$ and pick $R^{+}=\{v_5\}$ to take the place of the removed steps. Finally we get the random walk updated $R(t) = \{v_3, v_6, v_2, v_5\}$ and the estimates are updated according to the forementioned equations respectively.



At t the random walk may be updated as $R(t) = (v_3, v_6, v_2, v_5)$.

Fig. 2. An example of a random walk on a graph with an edge added

There's a little difference in the case of edge removal. It is supposed that e_{uv} is removed at time t. The proposed algorithm updates the stored random walk if and only if e_{uv} is passed by the previous random walk. Approach of updating the random walk and estimates is analogous with the case of edge addition.

3 Correctness and Complexity

3.1 Correctness

Here we prove the proposed algorithm is correct via comparing the proposed algorithm with the baseline algorithm on a same graph. We demonstrate that the set of random walks generated by the proposed algorithm on graph $G(t)$ is equal to the set of random walks generated by the baseline algorithm.

Proof. First, we prove that for any random walk $R(t)$ generated by the baseline algorithm on $G(t)$, there is an equal random walk $R(t)$' generated by the proposed algorithm on the same graph. There are two cases. In the first case, we suppose that $R(t)$ doesn't access either v_u or v_v. The proposed algorithm inherits the random walk $R(t-1)$ generated by baseline algorithm on $G(t-1)$. And we could always find a random walk path $R(t)' = R(t-1)$ which is equal to $R(t)$. In the second case, we consider situations that $R(t)$ accesses either v_u or v_v. Let's suppose that $R(t)$ accesses v_u at its kth step. Similar as the first case, we could always find a path $R(t-1)$ who is equal to $R(t)$ at the first k steps. At the $(k+1)$th step, it takes $1/d_u$ probability to access each neighbor of v_u (including v_v) in $R(t)$. At the $(k+1)$th step of $R(t)$', it takes $1/d_u$ probability to access v_v (rerouting $R(t-1)$ from the $(k+1)$th step) and $(d_u-1)/d_u$ probability to access one of v_u's neighbors except v_v (inheriting from $R(t-1)$ without rerouting). Thus it is easy to find a random walk $R(t)$' which is equal to $R(t)$ from the $(k+1)$th step to the end of the random walk path. In summary, for any random walk generated by the baseline algorithm there's always an equal random walk generated by the proposed algorithm on the same graph.

In a similar way, it is easy to prove that for any random walk generated by the proposed algorithm, there's an equal random walk generated by the baseline algorithm on the same graph. Thus, the sets of random walk path generated by the proposed algorithm and the baseline algorithm respectively are equal. So the proposed algorithm is correct, due to it is demonstrated that the baseline algorithm is correct in [9, 11].

3.2 Computing Complexity

Here we analysis the complexity of updating estimate of clustering coefficient as graph evolves all the time. Assuming e_{uv} is the edge added at time t and M_t is the number of random walks rerouted. Then we have $E[M_t] \leq \alpha n/m$, where $\alpha n = r$, the length of a random walk path.

Proof. It is intuitive that most edges in the graph are not accessed by the random walk. A random walk needs to be updated only if it accesses v_u (or v_v) at time $t-1$ and chooses v_v (or v_u) as the next step. For an arbitrary vertex v_u, the expectation of the number of times v_u is accessed by a random walk is $\pi_u r$. The probability that node v_u is accessed at each step of a random walk, donated by π_u, is equal to d_u/D, which is proved in [9].The probability of choosing v_v as the next step of v_u is $1/d_u$. Thus, consider the expectation of M_t, which can be expressed as Eq. (5). As a random walk updated, the amount of work is O(r) at most. In summary, we could get that the upper bound of expected amount of work that the proposed algorithm needs to do is $O((\alpha n)^2/m)$ as an arbitrary edge arrives randomly.

$$E[M_t] = r\sum_{i,j}\left(\pi_i/d_i + \pi_j/d_j\right)\Pr[i = u, j = v] \approx 2r\sum_{i}\left(d_i/d_i D\right)\Pr[i = u] = 2r/D = \alpha n/m \quad (5)$$

4 Evaluations

4.1 Experiment Setup

In experiments, we mainly compare the accuracy and performance of the proposed algorithm with the baseline algorithm [9], a state-of-art algorithm estimating the clustering coefficient via random walk. We implement both of the algorithms by Java. Our experiments all run on a machine with Intel(R) Core (TM) i7-2600 CPU and 8 GB RAM. The graphs used in the experiments are some public datasets downloaded from SNAP [12]. We deal with all of them as undirected graphs by ignoring the direction of directed edges. Table 1 lists the main characteristics of the graphs.

Table 1. Main parameters of data sets

Graph	Background	Nodes	Edges	a	g
amazon0601	Amazon product co-purchasing	403K	3.4M	0.4177	0.1656
as-Skitter	Internet topology	1.6M	11.1M	0.2581	0.0054
cit-Patents	Citation network among US Patents	3.7M	16.5M	0.0757	0.0671
com-DBLP	DBLP collaboration network	317K	1.0M	0.6324	0.3064
com-Youtube	Youtube online social network	1.1M	3.0M	0.0808	0.0062
higgs-twitter	Followers graph on Twitter	457K	14.9M	0.1887	0.0102
soc-Pokec	Pokec online social network	1.6M	30.6M	0.1094	0.0468
web-Google	Web graph from Google	875K	5.1M	0.5143	0.0552
wiki-Talk	Wikipedia talk network	2.4M	5.0M	0.0526	0.0033

For generating the evolving graphs, we randomly remove 100000 edges from each graph and use the rest part as the initial graph in our experiments. The initial estimates are computed by the baseline algorithm. Then we add the removed edges one by one and estimate the average and global clustering coefficient respectively via the proposed algorithm and its competitor. For all experiments, we set $r = 0.02n$ by default, which is enough for getting accurate approximations [9, 11]. Moreover, we run the experiments for 100 times on each graph and use the average results in our evaluations.

4.2 Accuracy

We use RMSE (Root Mean Square Error) to measure the accuracy, which is defined as Eq. (6), where c stands for the exact clustering coefficient and $\hat{c}$ stands for its estimate. We computed the exact clustering coefficient by a node-iteration based method [13]. Table 2 provides a comparison of the average RMSE.

$$RMSE = \sqrt{E\left[(\hat{c}/c - 1)^2\right]} \tag{6}$$

Table 2. Comparison of RMSE

Graph	Average clustering coefficient		Global clustering coefficient	
	Proposed	Baseline	Proposed	Baseline
amazon0601	0.0424	0.0424	0.0959	0.0912
as-Skitter	0.0477	0.0486	0.1928	0.1989
cit-Patents	0.0285	0.0297	0.0328	0.0313
com-DBLP	0.0385	0.0409	0.1373	0.1300
com-Youtube	0.0708	0.0808	0.1622	0.1649
higgs-twitter	0.0782	0.0790	0.1881	0.1920
soc-Pokec	0.0481	0.0512	0.0395	0.0390
web-Google	0.1025	0.1117	0.1125	0.5312
wiki-Talk	0.0399	0.0377	0.3526	0.3483

The results demonstrate that the proposed algorithm performs as well as the baseline algorithm in accuracy. The proposed algorithm achieves smaller RMSE in two-thirds of the experiments. RMSE of the proposed algorithm is about 12% smaller than the baseline algorithm in the best case and it is about 6% bigger than the baseline algorithm in the worst one. In summary, the proposed algorithm achieves close (or even smaller) RMSE comparing with the baseline algorithm for all experiments, which means the proposed algorithm performs as well as the baseline algorithm.

4.3 Performance

We measure the running time and the number of random walk rerouting of the proposed algorithm to evaluate the performance. We also define the speedup as the ratio of the running time of the baseline algorithm to the running time of the proposed algorithm. Table 3 provides a detailed comparison of the running time, the number of random walk rerouting and the speedup of the proposed algorithm on each graph.

Table 3. Comparison of the running time and number of times the random walk rerouted

Graph	Average clustering coefficient			Global clustering coefficient		
	Running time (s)	Number of rerouting	Speedup	Running time (s)	Number of rerouting	Speedup
amazon0601	1.97	28661	186.9	1.97	28865	187.9
as-Skitter	22.41	26250	356.2	23.43	26000	353.3
cit-Patents	22.80	37800	217.9	22.83	37456	216.4
com-DBLP	1.62	40080	209.6	1.74	40169	194.6
com-Youtube	12.52	43595	305.1	12.63	42915	180.6
higgs-twitter	6.20	6692	531.1	6.22	6825	524.1
soc-Pokec	6.41	13204	520.6	6.23	13144	549.1
web-Google	4.88	29991	163.9	5.02	30194	181.8
wiki-Talk	125.45	73340	129.5	125.88	73385	115.8

It is obvious that the proposed algorithm reduces the running time significantly for dealing with large and evolving graphs. The speedup of the proposed algorithm comparing with the baseline algorithm for dealing with graph with 100000 evolving edges is in the range of 115 to 549. We also find that estimating the average and global clustering coefficient on a graph via the proposed algorithm cost similar running time.

5 Conclusion

In this paper, we propose an incremental algorithm for tracking approximate clustering coefficient on dynamic graph via random walk method. As an arbitrary edge is added and/or removed, our algorithm replaces partial random walk path around the evolving part and updates the estimate based on previous result. The accuracy and performance improvement of the proposed algorithm are verified through analysis in theory and experiments on real-world graphs. It is demonstrated that the proposed algorithm improves the performance effectively comparing with the state-of-art algorithm based on random walk.

References

1. Watts, D.J., Strogatz, S.H.: Collective dynamics of 'small-world' networks. Nature **393** (6684), 440–442 (1998)
2. Shen, G., Gao, B., Liu, T.Y., Feng, G., Song, S., Li, H.: Detecting link spam using temporal information. In: 6th IEEE International Conference on Data Mining, pp. 1049–1053. IEEE Press, New York (2006)
3. Benevenuto, F., Rodrigues, T., Almeida, V., Almeida, J., Gonçalves, M.: Detecting spammers and content promoters in online video social networks. In: 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 620–627. ACM, New York (2009)
4. Akoglu, L., Dalvi, B.: Structure, tie persistence and event detection in large phone and SMS networks. In: 8th Workshop on Mining and Learning with Graphs, pp. 10–17. ACM, New York (2010)
5. Becchetti, L., Boldi, P., Castillo, C., Gionis, A.: Efficient algorithms for large-scale local triangle counting. ACM Trans. Knowl. Discov. Data (TKDD) **4**(3), 13 (2010)
6. Park, H.M., Chung, C.W.: An efficient mapreduce algorithm for counting triangles in a very large graph. In 22nd ACM International Conference on Information & Knowledge Management, pp. 539–548. ACM, New York (2013)
7. Tsourakakis, C.E., Kang, U., Miller, G.L., Faloutsos, C.: DOULION: counting triangles in massive graphs with a coin. In: 15th ACM SIGKDD International Conference on Knowledge Discovery and Data mining, pp. 837–846. ACM, New York (2009)
8. Seshadhri, C., Pinar, A., Kolda, T.G.: Wedge sampling for computing clustering coefficients and triangle counts on large graphs. Stat. Anal. Data Min. ASA Data Sci. J. **7**(4), 294–307 (2014)
9. Hardiman, S.J., Katzir, L.: Estimating clustering coefficients and size of social networks via random walk. In: 22nd International Conference on World Wide Web, pp. 539–550. ACM, New York (2013)

10. Costa, L.D.F., Rodrigues, F.A., Travieso, G., Villas Boas, P.R.: Characterization of complex networks: a survey of measurements. Adv. Phys. **56**(1), 167–242 (2007)
11. Katzir, L., Hardiman, S.J.: Estimating clustering coefficients and size of social networks via random walk. ACM Trans. Web (TWEB) **9**(4), 19 (2015)
12. Stanford large network dataset collection. http://snap.stanford.edu/data/index.html
13. Schank, T.: Algorithmic aspects of triangle-based network analysis. Ph.D. thesis, Universität Karlsruhe (TH) (2007)