An Incremental Algorithm for Estimating Average Clustering Coefficient Based on Random Walk

Qun Liao, Lei Sun, He Du and Yulu Yang

College of Computer and Control Engineering, Nankai University, Tianjin, China

{liaoqun, sunleier, duhe}@mail.nankai.edu.cn,
yangyl@nankai.edu.cn

Abstract. Clustering coefficient is an important measure in social network analysis, community detection and many other applications. However, it is expensive to compute clustering coefficient for the real-world networks, because many networks, such as Facebook and Twitter, are usually large and evolving continuously. Aiming to improve the performance of clustering coefficient computation for the large and evolving networks, we propose an incremental algorithm based on random walk model. The proposed algorithm stores previous random walk path and updates the the average clustering coefficient estimation through reconstructing partial path in an incremental approach, instead of recomputing clustering coefficient from scratch as long as graph changes. Theoretical analysis suggests that the proposed algorithm improves the performance of clustering coefficient estimation for dynamic graphs effectively without sacrificing in accuracy. Extensive experiments on some real-world graphs also demonstrate that the proposed algorithm reduces the running time significantly comparing with a state-of-art algorithm based on random walk.

Keywords: Clustering Coefficient; Graph Analysis; Incremental Algorithm; Random Walk

1 INTRODUCTION

Clustering coefficient [1] plays an important role in mining useful knowledge from real-world networks, such as the World Wide Web and online social networks. Clustering coefficient describes the homophily (people become friends with those similar to themselves) and transitivity (friends of friends become friends) of a network, which is widely used in community detection [2], spam detection [3], protein-protein interaction network analysis [4] and many other applications.

In many real-world applications the networks are large and evolving all the time. For example, Google crawls more than 600K new webpages every second [5] and the number of users of Facebook is over 1.6 billions and increases rapidly. Though many researches have optimized the performance of computing clustering coefficient based on static graph model, it is still expensive to compute clustering coefficient for large and evolving networks.

Aiming to improve the performance of clustering coefficient computation for dynamic graphs, we propose an incremental algorithm in this paper. It computes clustering coefficient via random walk initially. As the graph changes, it reuses previous random walk and gets result updated via reconstructing random walk path incrementally, instead of recomputing from scratch. Theoretical analysis and experimental evaluations both demonstrate that the proposed algorithm improves the performance effectively.

2 PRELIMINARY AND PRIOR WORKS

The clustering coefficient was first introduced in [1] to describe the degree of how nodes are closed to their neighbors. In this paper, we focus on the average clustering coefficient [6], a widely used version of clustering coefficient.

Let $G = (V, E)$ be an undirected connected graph with n nodes and m edges. We donate by v_i a node in G and by d_i the degree of node v_i. We define $D = \sum_{i=1}^{n} d_i$. The adjacency matrix, donated by A, is a $n \times n$ symmetric matrix, $A_{ij} = A_{ji} = 1$ if and only if there is an edge between v_i and v_j, and $A_{ij} = A_{ji} = 0$ otherwise. For any node v_i, $A_{ii} = 0$.

If $A_{ij} = A_{ik} = 1$, and $j < k$, the triplet (v_j, v_i, v_k) is defined as a wedge. For any wedge (v_j, v_i, v_k), if $A_{jk} = 1$, we define it a triangle, denoted as $<v_j, v_i, v_k>$. For any node v_i, the number of triangles $<v_j, v_i, v_k>$ is donated by l_i. It is obvious that l_i is equal to the number of edges between v_i's neighbors.

The local clustering coefficient of any node v_i is donated by c_i. For node v_i where $d_i > 1$, we define c_i as $2l_i/d_i(d_i-1)$. For node v_i where $d_i \leq 1$, c_i is defined as 0. The average clustering coefficient, donated by c_l, is the average of local clustering coefficient over the set of nodes, which is defined as Eq. (1).

$$c_l = \frac{1}{n} \sum_{i=1}^{n} c_i \tag{1}$$

It is expensive to compute c_l for a large graph due to counting the number of triangles is a challenging task. The best known algorithm for exact triangles counting requires $O(m^{2\omega/(\omega+1)})$ time [7], where $\omega < 2.376$ is the exponent of matrix multiplication [8]. However, it is not practical for large graphs due to its considerable cost of memory.

Random sampling based algorithms [9, 10] are preferable because an accurate approximation is sufficient in a large amount of real-world applications. Among these algorithms, the random walk based algorithm [9] performs well in both accuracy and efficiency. Its another advantage is that it only relies on external access to the graph and requires no prior knowledge of the graph (e.g. the number of nodes). It makes the algorithm adapt to the real-world social network analysis well, because for many online social networks it is not realistic to access the entire graph or another extra information for security and performance reasons.

A main limitation of the work in [9] is that it deals with graphs statically. So it is still expensive to deal with dynamic graphs due to the global recomputation as graph changes. Though streaming algorithms, such as Buriol et al. [11] and Becchetti et al. [12], estimate the clustering coefficient in a streaming model, these algorithms require to access each edge once at least, which limits their performance and make them inefficient comparing to the algorithms based on random walk, which only access a small number of edges among the whole graph.

3 INCREMENTAL ALGORITHM

3.1 Method

To improve the performance of computing clustering coefficient for dynamic graphs, we proposed an incremental algorithm based on the idea of reusing as much previous results as possible. The proposed algorithm stores previous estimated clustering coefficient and random walk path. It gets the estimation updated as graph evolves via reconstructing partial of the stored random walk path. We suppose that the average clustering coefficient is computed initially by the algorithm in [9], and we call the algorithm in [9] as baseline algorithm in the rest of this paper. The baseline algorithm generates a random walk with *r* steps, $R = (x_1, x_2, . . . , x_r)$ representing the random walk path, and estimates c_l according to $\hat{c}_l = \Phi_l / \Psi_l$, where $\hat{c}_l$ is the estimation, Φ_l and Ψ_l are defined as Eq. (2) and Eq. (3). And the variable φ_k is defined as Eq. (4).

$$\Phi_l = \frac{1}{r-2} \sum_{k=2}^{r-1} \phi_k \left(d_{x_k} - 1 \right)^{-1} \tag{2}$$

$$\Psi_l = \frac{1}{r} \sum_{k=1}^{r} \left(d_{x_k} \right)^{-1} \tag{3}$$

$$\varphi_k = A_{x_{k-1}, x_{k+1}}, 2 \le k \le r-1 \tag{4}$$

As an arbitrary edge e_{uv} added or removed, the proposed algorithm finds the set of positions where *u* and *v* emerging in the random walk path. We donate by *X* the set of positions of *u* in the random walk path, $X = \{x \mid$ the *x*th entry of *R* is $u\}$. Similarly, we define the set of positions of *v*, donated by *Y*. Then we compute the position where we start to replace the random walk path. We define two nonnegative integers z_1 and z_2. z_1 is the minimum entry in both *X* and *Y* and z_2 is the maximum number in these two sets. $z_1 = \min \{ z | z \in X \cup Y \}$ and $z_2 = \max \{ z | z \in X \cup Y \}$.

If z_1 equals to z_2, we compare z_1 and *r*, the length of the stored random walk path. If z_1 is greater than *r*/2, we start replacing the random walk path from z_1 to the end of the random walk path. Otherwise, we start from z_1 and replace the random walk path backward to its beginning. If z_1 and z_2 are not equal, we compute the summation of them. If their summation is smaller than *r*, we replace the random walk path from z_2 to the end. Otherwise, we start our replacement from z_1 backward to the beginning. Fig. 1 sketches how the stored random walk updated in different situations.

If *X* and *Y* are both empty, the proposed algorithm returns previous result directly without any recomputation. As the steps in the random walk path replaced, we also update φ_k and d_k correspondingly and recompute Φ_l and Ψ_l according to Eq. (2) and Eq. (3) respectively. So we get the estimation of clustering coefficient updated eventually.

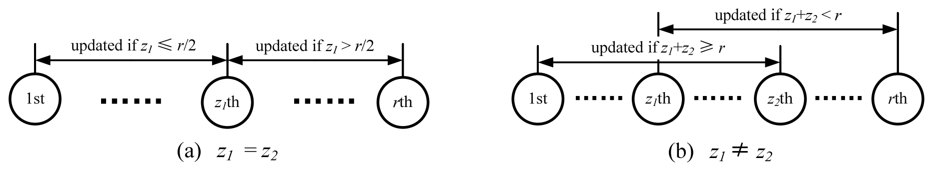

Fig. 1. Random walk updated according to the relations between r, z_1 and z_2

3.2 Correctness

It is instinctive to explain why the proposed incremental algorithm works. So we only give a simple version of explanation due to limitations on space. Supposing an arbitrary edge changes, there are only two situations: 1) the stored random walk path passes the endpoints of the changed edge; 2) the stored random walk path misses the endpoints.

In the first situation, the proposed algorithm updates random walk around the newly changed graph, thus it is equivalent to computing the clustering coefficient via the baseline algorithm on the newly changed graph.

In the second situation, the random walk doesn't get involved in the changing of graph, which is equivalent to computing the clustering coefficient via the baseline algorithm on the newly changed graph without accessing the newly changed edges. It is supposed there is only an edge changed at a time point, which takes a very tiny percentages of the millions of edges. The length of random walk is always relatively short, only takes 1 or 2 percentages of the number of nodes. Therefore, it is in a quite large probability that the baseline algorithm gets estimations updated for the newly changed graph without accessing the endpoints of the changed edge.

Consequently, the proposed incremental algorithm is correct as long as the baseline algorithm is correct. A detailed proof of the correctness of the baseline algorithm is provided in [9]. So we could confirm that the proposed algorithm is correct.

3.3 Computing Complexity

The main computation of the proposed algorithm is reconstructing the stored random walk path and updating Φ_l and Ψ_l. Thus the amount of computation of the proposed algorithm is linear to the number of steps of the stored random walk path rerouted.

It is intuitive that most edges in the graph are not accessed in the random walk. As e_{uv} arriving at time t, a random walk is needed to be updated only if it passes through either u or v. The probability that node u is accessed by a random walk, donated by π_u, is equal to d_u/D, which is proved in [9]. Define M to be the number of random walk path needed to be updated as an arbitrary e_{uv} changes, we get its expectation $E[M]$ as Eq. (5). Because the upper bound of amount of steps needed to be replaced is r. Hence, the amount of expected work as a randomly picked edge changes is O(mr^2) at most.

$$E[M] \le \sum_{e_{uv}\in E} r\left(\pi_{u_t} + \pi_{v_t}\right)\Pr\left[u_t = u, v_t = v\right] \approx 2mr\frac{\sum_{u\in V} d_u}{D} = 2mr\frac{n\bar{d}}{2m} = mr \tag{5}$$

4 EVALUATION

4.1 Experiment Setup

We evaluate the accuracy and performance of our algorithm via comparing with the algorithm in [9] as baseline. We implement both the algorithms in Java and run our experiments on a machine with Intel(R) Core(TM) i7-2600 CPU@3.80GHz and 8GB of RAM. We use some real-world graphs downloaded from SNAP [13]. The key parameters of these graphs are listed in Table 1.

Table 1. Main parameters of data sets

Graph	Background	Nodes	Edges	Average Clustering Coefficient
com-Amazon	product network	335K	926K	0.3967
com-DBLP	collaboration network	317K	1050K	0.6324
web-Stanford	Web graph	282K	2312K	0.5976

We use graphs with edges added to simulate evolving graphs in our experiments. We randomly remove a certain number (e.g. 1000, 2000, ..., 10000) of edges in each graph and use the rest part as the initial graph. We compute the average clustering coefficient on the initial graph via the baseline algorithm. Then we add the removed edges one by one and update the estimations via the proposed algorithm and its competitor. For both algorithms, we set $r = 0.02n$ by default, which is enough for accurate estimations [9].

4.2 Accuracy

We use RMSE, defined as Eq. (6), to evaluate the accuracy. We run each experiment for 1000 times and compute the average RMSE in our evaluation.

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

The comparison of RMSE between our algorithm and the baseline algorithm is depicted as Fig. 2, where the dotted lines show the average RMSE among experiments with different number of added edges. Smaller RMSE means performing better in accuracy.

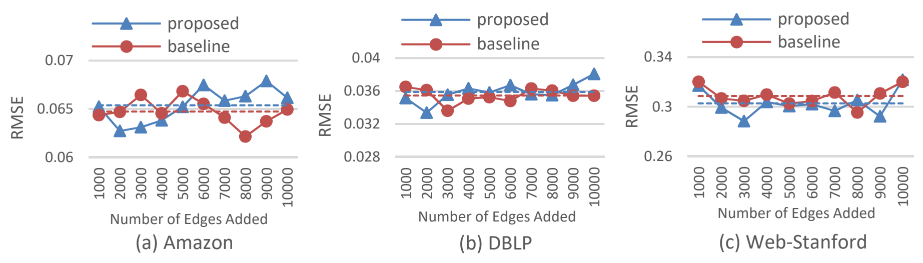

Fig. 2. Comparison of RMSE

We find that for all experiments, our algorithm achieves close (or even smaller) RMSE as its non-incremental competitor. The average RMSE of the proposed algorithm is larger about 1% (on Amazon and DBLP) and smaller about 2% (on Web-Stanford) than the baseline algorithm. The results demonstrate that the accuracy of our algorithm is in the same order of the baseline.

We also evaluate the effects of r, the length of the random walk path, on the accuracy of our algorithm. We run experiments with $r = 0.01n$ and $r = 0.02n$ respectively. A distinct comparison of the accuray of experiments with different r is depicted as Fig. 3. It is clear that as the rise of r, RMSE decreases distinctly. More precisely, as r increases as much as 200% and RMSE decreases by over 28% at most.

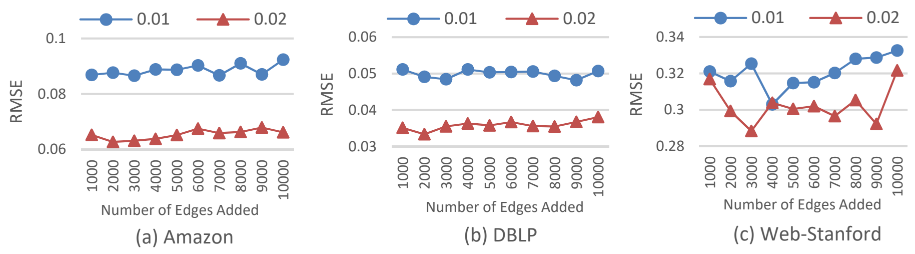

Fig. 3. Comparison of RMSE with different length of random walk path

4.3 Performance

To evaluate performance of our algorithm, we compare the running time on each graph. We find that our incremental algorithm runs much faster than the baseline algorithm. To depict the performance improvement directly, we present the speedup ratio of the incremental algorithm to the baseline algorithm in Fig. 4, where the dotted lines present the trends of speedup ratio as the number of added edges increasing.

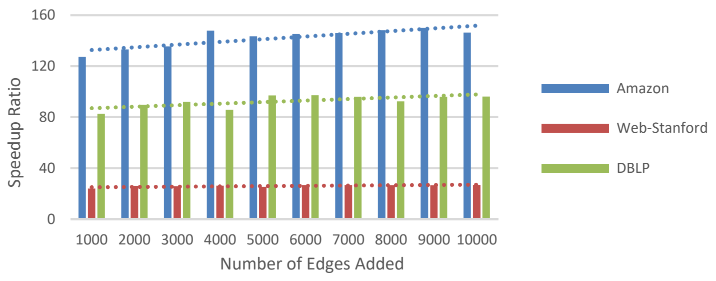

Fig. 4. Speedup ratio of the proposed algorithm

From the results we find that the proposed algorithm improves the performance of estimating clustering coefficient for dynamic graphs significantly. Comparing to the baseline algorithm, it speeds up the computation over 20 times around all experiments. Moreover, it speeds up the computation more than 140 times at the best cases (on

Amazon). The trends of the speedup ratios among all experiments also provide an obvious rise as the amount of added edges increasing. It implies that the proposed algorithm performs better for graph with a larger number of changed edges.

The comparison of running time of experiments on different graphs is shown as Fig. 5 (a). It depicts the running time of the proposed algorithm on each graphs for 1000 times respectively. We find that the running time of the proposed algorithm is almost linear with the number of edges. Moreover, the running time of experiments is roughly proportional to the number of edges of the graph. The running time on Web-Stanford dataset is about 3.2 times that of the runing time on DBLP, and almost 4.9 times that of the runing time on Amazon. We also count the number of times of the endpoints of the changed edges emerging in the stored path. As Fig. 5 (b) depicted, the counted number is linear with O(rm), which supports our analysis above.

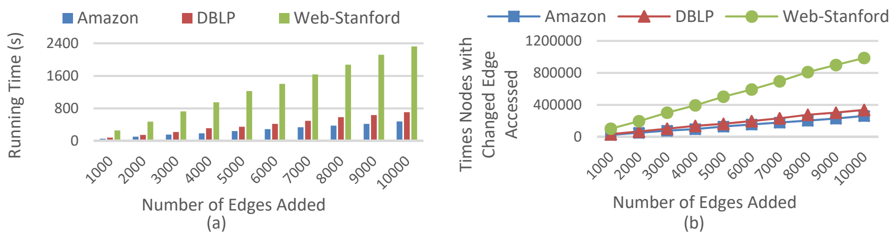

Fig. 5. Performance of the proposed algorithm

We run experiments with $r = 0.01n$ and $r = 0.02n$ respectively, to evaluate the effects of r on the performance of the proposed algorithm.We find that the running time is about linear to r^2 on each graph, depicted as Fig. 6. It meets our analysis in theory. As there are 10000 edges inserted, the running time of the proposed algorithm with $r = 0.02n$ is about 4 times that of experiment with $r = 0.01n$. The result is intuitive, a longer random walk means a higher probability of the changed edges accessed by the random walk and a bigger amount of computation as a random walk updated.

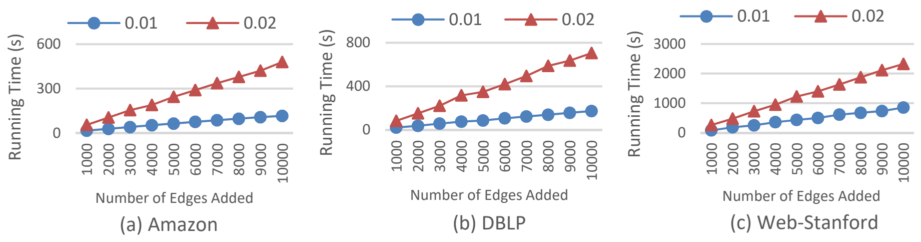

Fig. 6. Comparison of running time with different length of random walk path

5 CONCLUSION

In this paper, we propose an incremental algorithm for estimating the average clustering coefficient for dynamic graphs based on random walk. As an arbitrary edge is added and/or removed, the proposed algorithm replaces partial random walk path around the evolving part and updates the estimation based on previous result. The performance improvement of the proposed algorithm is verified through analysis in theory and experiments on real-world graphs. Comparing with a state-of-art algorithm based on random walk, the proposed algorithm improves the performance effectively without sacrifices in accuracy.

6 REFERENCES

1. Watts, D. J., Strogatz, S. H.: Collective dynamics of 'small-world' networks. nature, 393(6684), 440-442(1998). doi: 10.1038/30918
2. Zhang, P., Wang. J., Li. X.: Clustering coefficient and community structure of bipartite networks. Physica A: Statistical Mechanics and its Applications, 387(27), 6869-6875(2008). doi: 10.1016/j.physa.2008.09.006
3. Xu, Q., Xiang, E. W., Yang, Q.: Sms spam detection using noncontent features. IEEE Intelligent Systems, 27(6), 44-51(2012). doi: 10.1109/MIS.2012.3
4. Stelzl, U., Worm, U., Lalowski, M.: A human protein-protein interaction network: a resource for annotating the proteome. Cell, 122(6), 957-968(2005)
5. Google Inside Search. https://www.google.com/intl/ta/insidesearch/howsearch-works/thestory/index.html
6. Costa, L. F., Rodrigues, F. A., Travieso, G.: Characterization of complex networks: A survey of measurements. Advances in physics, 56(1), 167-242(2007). doi:10.1080/00018730601170527
7. Alon, N., Yuster, R., Zwick, U.: Finding and counting given length cycles. Algorithmica, 17(3), 209-223(1997)
8. Coppersmith, D., Winograd, S.: Matrix multiplication via arithmetic progressions. Journal of symbolic computation, 9(3), 251-280(1990)
9. Katzir, L., Hardiman, S. J.: Estimating clustering coefficients and size of social networks via random walk. ACM Transactions on the Web (TWEB), 9(4), 19(2015). doi: 10.1145/2790304
10. Schank, T., Wagner, D.: Approximating clustering-coefficient and transitivity. Universität Karlsruhe, Fakultät für Informatik(2004)
11. Buriol, L. S., Frahling, G., Leonardi, S.: Counting triangles in data streams. In: 25th ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, pp. 253-262. ACM, New York (2006). doi:10.1145/2902251.2902283
12. Becchetti, L., Boldi, P., Castillo, C.: Efficient semi-streaming algorithms for local triangle counting in massive graphs. In: 14th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 16-24. ACM, New York (2008). doi: 10.1145/1401890.1401898
13. Stanford Large Network Dataset Collection. http://snap.stanford.edu/data/index.html