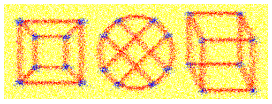
Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 9, no. 2, pp. 265–275 (2005)

Approximating Clustering Coefficient and Transitivity

Thomas Schank *Dorothea Wagner*

Algorithmics
Department of Computer-Science
University of Karlsruhe, Germany
http://i11www.ira.uka.de/algo/
schank|dwagner@ira.uka.de

Abstract

Since its introduction in the year 1998 by Watts and Strogatz, the clustering coefficient has become a frequently used tool for analyzing graphs. In 2002 the transitivity was proposed by Newman, Watts and Strogatz as an alternative to the clustering coefficient. As many networks considered in complex systems are huge, the efficient computation of such network parameters is crucial. Several algorithms with polynomial running time can be derived from results known in graph theory. The main contribution of this work is a new fast approximation algorithm for the weighted clustering coefficient which also gives very efficient approximation algorithms for the clustering coefficient and the transitivity. We namely present an algorithm with running time in $\mathcal{O}(1)$ for the clustering coefficient, respectively with running time in $\mathcal{O}(n)$ for the transitivity. By an experimental study we demonstrate the performance of the proposed algorithms on real-world data as well as on generated graphs. Moreover we give a simple graph generator algorithm that works according to the preferential attachment rule but also generates graphs with adjustable clustering coefficient.

Article Type	Communicated by	Submitted	Revised
concise paper	G. Di Battista	June 2004	October 2005

The authors gratefully acknowledge financial support from DFG under grant WA 654/13-2 and from the European Commission within FET Open Project COSIN (IST-2001-33555).

1 Introduction

Recently, there is growing interest in understanding the structure, dynamics and evolution of large networks like the Internet, the World Wide Web, technological and biological networks or social networks. One way of analyzing specific properties of networks consists in computing and comparing certain local or global network indices. Algorithmic aspects in network analysis concern the correct and fast computation of those. Vertex indices are often easily computable in polynomial time. However, if networks are large even polynomial running times that are e.g. cubic in the number of nodes are not acceptable. Under some circumstances even sublinear algorithms are desired. A frequently used tool for analyzing graphs is the clustering coefficient introduced in [13] respectively the transitivity proposed in [12]. This paper concentrates on the algorithmic aspects of computing those indices. The main contribution of this work is a constant time approximation algorithm for the clustering coefficient. We also present a generator algorithm that follows the linear preferential attachment principle and produces graphs with the desired clustering coefficient. Such generators have been proposed before but, they either only work for a limited set of parameters as in [10], or they are quite complicated as in [11]. An experimental study demonstrates the performance of the proposed algorithms on real-world data as well as on generated graphs. These results also support the assumption that the values of the clustering coefficient and the transitivity differ in general.

2 Definitions and Properties

2.1 Basic Definitions

Let $G = (V, E)$ be an undirected, simple (no self-loops, no multiple edges) graph (network) with a set of nodes (vertices) V and a set of edges E. We use the symbol n for the number of nodes and the symbol m for the number of edges. The *degree* $d(v)$ of node v is defined to be the number of nodes in V that are adjacent to v. A complete subgraph of three nodes of G can be considered as a *triangle* Δ. Motivated by this the *number of triangles of node* v is defined as $\delta(v) = |\{\{u, w\} \in E : \{v, u\} \in E \text{ and } \{v, w\} \in E\}|$. Since each triangle contains three nodes (hence if we sum over all nodes each triangle is counted three times) the following definition is intuitive

$$\delta(G) = 1/3 \sum_{v \in V} \delta(v). \tag{1}$$

A *triple* Υ at a node v is a path of length two for which v is the center node. The number of *triples of node* v is then defined as $\tau(v) = \binom{d(v)}{2}$, and summing the triples of all nodes defines

$$\tau(G) = \sum_{v \in V} \tau(v). \tag{2}$$

2.2 Clustering Coefficient

The clustering coefficient was introduced by Watts and Strogatz [13] in the context of social network analysis. Given three actors u, v and w with mutual relations between v and u as well as between v and w, it is supposed to represent the likeliness that u and w are also related. With the introduced notation the *clustering coefficient* for a node v with $d(v) \geq 2$, is defined as $c(v) = \delta(v)/\tau(v)$. The clustering coefficient $C(G)$ of a graph G is the average over the clustering coefficients of its nodes

$$C(G) = \frac{1}{|V'|} \sum_{v \in V'} c(v), \tag{3}$$

where V' is the set of nodes v with $d(v) \geq 2$. Alternatively to Eq. 3 the term $c(v)$ is sometimes defined to be either zero or one for nodes v with degree at most 1 and included in the sum. However, the choice of the definition is important as can be seen from the results of our experiments in Sec. 5.

2.3 Transitivity

The *transitivity* of a graph G as used in [12] is defined as

$$T(G) = \frac{3\,\delta(G)}{\tau(G)}. \tag{4}$$

It is essentially the *transitivity ratio* [8] restricted to undirected graphs. The transitivity as in Eq. 4 was claimed to be equal to the clustering coefficient as in Eq. 3 by Newman, Watts and Strogatz [12]. However this is clearly not true and the complete graph of 4 nodes with one edge removed is the smallest counterexample [4]. A formal relation between the two parameters will follow in the next section. From now on we simply use C, T, δ and τ if the corresponding graph or node is clear from context.

2.4 Generalized Clustering Coefficient

For a weight function $w : V \to \mathbb{R}^+$ we define the *weighted clustering coefficient* to be

$$C_w(G) = \frac{1}{\sum_{v \in V'} w(v)} \sum_{v \in V'} w(v)c(v). \tag{5}$$

Two implicit weight functions are immediate, the degree-based weight $w(v) = d(v)$, and the weight given by the number of triples $w(v) = \tau(v)$. In the second case, $\tau(v)$ simply cancels out in each additive term of the numerator, and we get $C_\tau(G) = \sum \delta(v) / \sum \tau(v)$ which can be rewritten with equation Eq. 1 and Eq. 4 to

$$T(G) = C_\tau(G) \tag{6}$$

and we recognize the transitivity as the triple weighted clustering coefficient. An equivalent formulation of Eq. 6 is presented in [4]. The following properties are immediate.

Corollary 1 *The indices C and T are equal for graphs where*

- *all nodes have the same degree, or*
- *all nodes have the same clustering coefficient.*

The first property is quite interesting with respect to the small-world networks of Watts and Strogatz [13] where node degrees do not differ much. Actually in [3] an 'alternative formulation' for C is introduced, claiming that it does not alter the 'physical significance' for certain generated small world networks. This 'alternative formulation' turns out to be equivalent to the transitivity T.

2.5 Generalized Transitivity

Let Π be the set of all triples in a graph G, then $\tau(G) = |\Pi|$. Further consider the mapping $X : \Pi \to \{0, 1\}$, where $X(\Upsilon)$ equals one if there is an edge between the outer nodes of the triple Υ, and zero otherwise. Then we can rewrite the transitivity as

$$T(G) = \frac{1}{|\Pi|} \sum_{\Upsilon \in \Pi} X(\Upsilon).$$

This equation is similar to the definition of the clustering coefficient in Eq. 3. Again we can consider a weight function $\varpi : \Pi \to \mathbb{R}^+$ and define

$$T_\varpi(G) = \frac{1}{\sum\limits_{\Upsilon \in \Pi} \varpi(\Upsilon)} \sum_{\Upsilon \in \Pi} \varpi(\Upsilon) X(\Upsilon). \tag{7}$$

Lemma 1 *The weighted clustering coefficient is a special case of the weighted transitivity.*

Proof: For a node weight function w, let $\varpi(\Upsilon) = \frac{w(v)}{\tau(v)}$ where v is the center of triple Υ. Further, let Π_v be the set of triples with center v. Then accordingly $\tau(v) = |\Pi_v|$ and $\sum_{\Pi_v} X(\Upsilon) = \delta(v)$. By a simple transformation we get $C_w = T_\varpi$ which completes the proof. □

The above proof will be useful for the approximation algorithms in Section 3. Note that for $w(v) = \tau(v), \forall v$, i.e. $\varpi \equiv 1$, $T_\varpi = T$ and for $w \equiv 1$, i.e. $\varpi(\Upsilon) = 1/\tau(v)$, $T_\varpi = C$, where again v is assumed to be the center of triple Υ.

3 Algorithms

We have seen that computing C and T involves the computation of the number of triangles. If we assume that $w(v)$ is computable in $\mathcal{O}(1)$ time, which we will do from now on, C_w is bounded by the complexity of counting triangles. Triangle counting is a well studied problem in graph theory. A very simple approach is to traverse over all nodes and check for existing edges between any pair of neighbors. This algorithm, which we call NODE-ITERATOR has running

time $\mathcal{O}\left(\sum_{v\in V}\binom{d(v)}{2}\right) \subset \mathcal{O}(nd_{\max}^2)$. The currently asymptotically most efficient algorithm with respect to the input size $n+m$ of the graph has running time $\mathcal{O}(m^{2\gamma/(\gamma+1)}) \subset \mathcal{O}(m^{1.41})$ [1], where $\gamma \leq 2.376$ [5] is the fast matrix multiplication exponent. In the following we will give linear respectively sublinear algorithms to approximate C_w, T and C.

3.1 Approximating C_w

Roughly speaking our approximation algorithm samples triples with appropriate probability. It then checks whether an edge between the non center nodes of the triple is present. Finally, it returns the ratio between the number of existing edges and the number of samples. The pseudo code is presented in Algorithm 1. We restrict the node weights to the strictly positive integers for simplicity.

Algorithm 1: C_w-Approximation

Input: integer k; array $A_{1,\dots,|V'|}$ of nodes $V' = \{v \in V\colon d(v) \geq 2\}$
node weights $w : V' \to \mathbb{N}_{>0}$; adjacency array for each node
Output: approximation of C_w
Data: node variables: u, w; integer variables: r, l, j, $W_{0,\dots,|V'|}$

$W_0 \leftarrow 0$
1 **for** $i = (1, \dots, |V'|)$ **do**
  $W_i \leftarrow W_{i-1} + w(A_i)$
$l \leftarrow 0$
2 **for** $i \in (1, \dots, k)$ **do**
3 $r \leftarrow$ `UniformRandomNumber(` $\{1, \dots, W_{|V'|}\}$`)`
4 $j \leftarrow$ `FindIndex(` $j : W_{j-1} < r \leq W_j$`)`
5 $u \leftarrow$ `UniformRandomAdjacentNode(`A_j`)`
  repeat
    $w \leftarrow$ `UniformRandomAdjacentNode(`A_j`)`
  until $u \neq w$
6 **if** *EdgeExists*(u, w) **then**
    $l \leftarrow l + 1$
return l/k

Theorem 1 *For a graph G with given node weights $w(v)$, a value $C_w^{approx}(G)$ that is in $[C_w(G) - \epsilon, C_w(G) + \epsilon]$ with probability at least $\frac{\nu-1}{\nu}$ can be computed in time $\mathcal{O}\left(n + \frac{\ln \nu}{\epsilon^2} \ln n\right)$.*

Proof: We prove that Algorithm 1 has the requested properties. Let us first consider the time complexity. The running time of the first for loop (starting at line 1) is obviously in $\mathcal{O}(n)$. For the second for loop (line 2), the *FindNode* function (line 4) can be executed in $\ln n$ steps by performing binary search. Choosing two adjacent nodes (line 5 to line 6) is expected to be in constant time. The *EdgeExists* function (line 6) is expected to be in constant time as

well if an efficient hash data structure is used for the edges. Finally, defining $k = \lceil \ln(2\nu)/(2\epsilon^2) \rceil$ gives total expected running time of $\mathcal{O}\left(\frac{\ln \nu}{\epsilon^2} \ln n\right)$ for the second for loop.

In order to prove the correctness for our choice of k, we make use of Hoeffding's bound [9]. Let X_i be independent real random variables bounded by $0 \leq X_i \leq M$ for all i. With k denoting the number of samples, and ϵ some error, the bound states:

$$\Pr\left(\left|\frac{1}{k}\sum_{i=1}^{k} X_i - \mathrm{E}\left[\frac{1}{k}\sum_{i=1}^{k} X_i\right]\right| \geq \epsilon\right) \leq 2e^{\frac{-2k\epsilon^2}{M^2}} \tag{8}$$

We have to prove that the expectation $\mathrm{E}(l/k)$ is equal to C_w and that the bounds on ϵ and $\frac{\nu-1}{\nu}$ are fulfilled for our choice of k. The proof of Lemma 1 implies that C_w can be computed by testing for each triple whether it is contained in a triangle or not. With the same notation as in Sec. 2.5 and particularly as in the proof of Lemma 1, we get

$$C_w(G) = \frac{1}{\sum\limits_{u \in V'} w(u)} \sum_{v \in V'} \sum_{\Upsilon \in \Pi_v} \frac{w(v)}{\tau(v)} X(\Upsilon)$$

where $w(v)/\tau(v)$ is the weight of the corresponding triple. This corresponds to the probability of $w(v)/\left(\tau(v)\sum w(u)\right)$ that a triple is chosen in one single loop starting at line 2. Hence, by linearity of the expectation $\mathrm{E}(l/k) = C_w$. The random variable $X(\Upsilon)$ is a mapping from Π to $\{0, 1\}$, and consequently $M = 1$. We can now immediately see that the bounds on ϵ and the probability $\frac{\nu-1}{\nu}$ are fulfilled for our choice of k. □

One may regard the error bound ϵ and the probability ν as fixed parameters. The number of triples $\tau(v)$ can be computed in $\mathcal{O}(1)$ time, if e.g. adjacency arrays are used. With this assumptions, we get immediately the following corollary.

Corollary 2 *Under the above assumptions the weighted clustering coefficient C_w and particularly the transitivity T can be approximated in $\mathcal{O}(n)$ time.*

3.2 Constant Time Approximation of C

If m grows more than linearly in n we have achieved a sublinear algorithm. However, to approximate C an even further improved bound can be achieved with some modifications of Algorithm 1: The first for loop (starting at line 1) and line 3 is removed. The node j (line 4) is now sampled uniformly from all nodes in V'.

Theorem 2 *The clustering coefficient C can be approximated in constant time.*

To see the correctness one simply verifies easily that a triple at center node v is sampled with probability $1/(\tau(v)|V'|)$ which corresponds to the correct weight $w \equiv 1$ or $\varpi = 1/\tau$ for obtaining C (compare to the last paragraph of Sec. 2.5). The rest follows analogously as in the proof of Theorem 1. Note, that the clustering coefficient can also be approximated with $\Theta(\log n)$ samples using standard Chernoff bounds [6].

4 A Preferential Attachment Graph Generator with Adjustable Clustering Coefficient

Algorithm 2: Graph Generator

Input: initial graph G: two connected nodes
integer: $n \geq 3$, $d \geq 2$, o

Output: graph G

for $i = (3, \ldots, n)$ **do**
  $v \leftarrow$ `NewNode()`
  for $1, \ldots, Min(i-1, d)$ **do**
    repeat
      $u \leftarrow$ `RandomNode(`*with prob* $d_u / \sum_V d(v)$`)`
    until *not* `EdgeExists(`v, u`)`
    `AddEdge(`v, u`)`
  for $1, \ldots, o$ **do**
    $u \leftarrow$ `RandomAdjacentNode(`v`)`
    repeat
      $w \leftarrow$ `RandomAdjacentNode(`v`)`
    until $w \neq u$
    if *not* `EdgeExists(`u, w`)` **then**
      `AddEdge(`u, w`)`

We give a graph generator algorithm that works according to the preferential attachment rule but also generates graphs with adjustable clustering coefficient. The sketch of a preferential attachment generator was introduced in [2] mainly to achieve graphs with the so called power law degree distribution found in many real networks [7]. However, the approach in [2] does not achieve the relatively high clustering coefficients found in many networks. Several methods were proposed to amend this. The generator presented in [10] is quite simple. However, high values for C are only verified for a restricted class of preferential attachment graphs, i.e., for the case where exactly 3 edges are added in every so called PA step. The authors expect that 'other values should give qualitatively the same behavior'. However, this is not confirmed by our preliminary experiments (not presented here for brevity). In contrast the approach proposed in [11] is

quite complicated. Additionally it functions by post-processing a given graph and hence does not model a growing process. The pseudo code of Algorithm 2

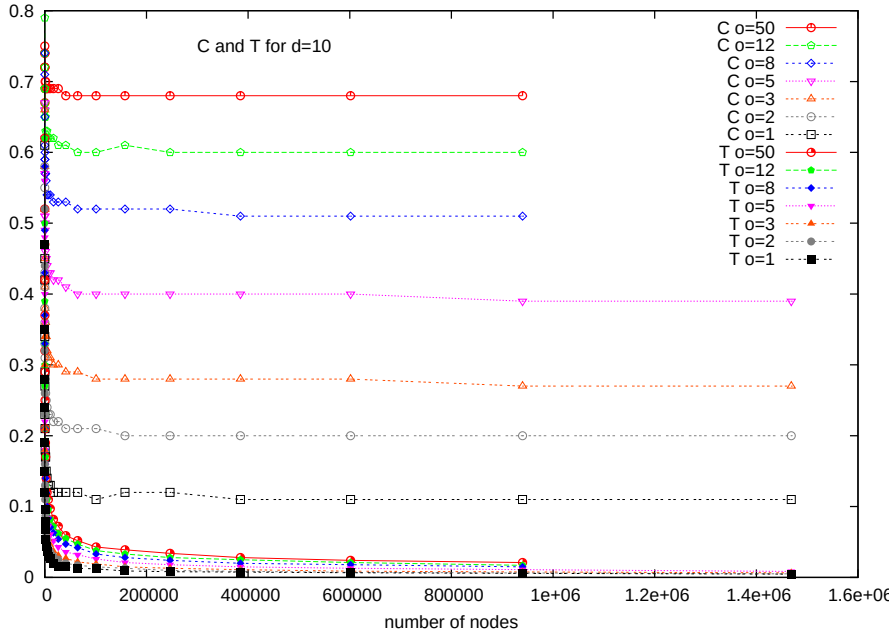

Figure 1: C and T for the generator with $d = 10$ and several o

presents a generator that is very simple. It runs in linear time with respect to the output size $n + m$ for fixed parameters d and o. The clustering coefficient of the output graph is adjustable, see Fig. 4. Algorithm 2 serves mainly as a generator to be used in the next section where the algorithms are experimentally evaluated.

5 Experiments

This section demonstrates the efficiency of the proposed algorithms in execution time. We compare an implementation APPROX of Algorithm 1 with running time $\mathcal{O}(n)$ to an implementation of NODE-ITERATOR, with running time in $\mathcal{O}\big(nd_{\max}^2\big)$ and an implementation of a algorithm $\text{AYZ}_{3/2}$ derived from the algorithm in [1]. In this case we did not use fast matrix multiplication for V_{high} as described in [1] but the standard method NODE-ITERATOR. The running time of $\text{AYZ}_{3/2}$ is therefore in $\mathcal{O}\big(m^{3/2}\big)$. The parameters of APPROX were set to $\epsilon = 0.001$ and $\nu = 0.5 \cdot 10^{10}$, throughout. The implementations were programmed in c++ and compiled with the GNU-compiler version 3.3. The experiments were carried out on a 2.4 GHz Intel-Xeon running Linux as the operating system. Figure 2 shows

the execution times for a series of graphs generated with GEN, an implementation of Algorithm 2. Table 5 shows results for two real networks. The first one is based on the movie actor database 2002 and the second corresponds to the autonomous system graph of the Internet.

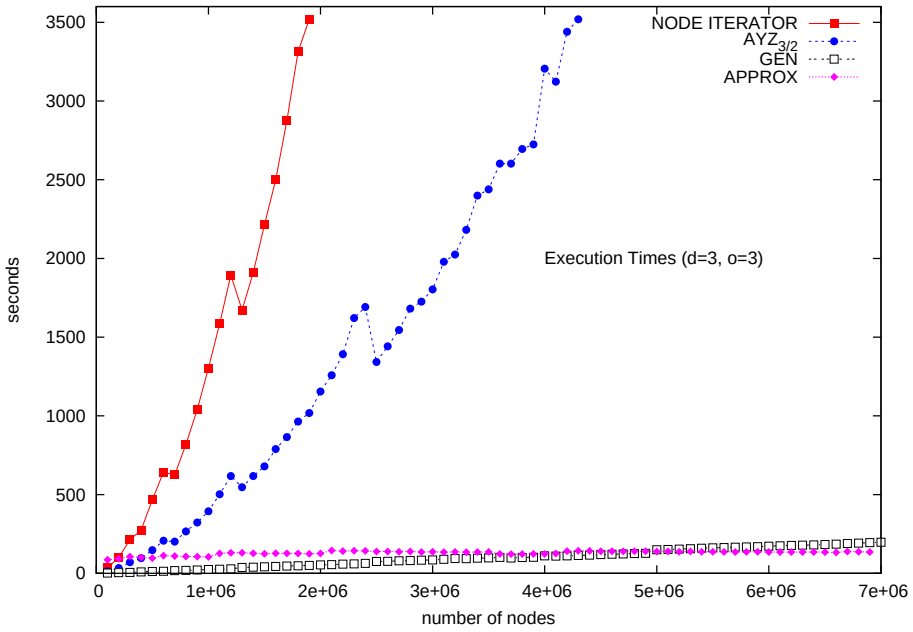

Figure 2: Execution Times

	Movie Actor	AS-Graph
n	$382 \cdot 10^3$	13164
m	$15 \cdot 10^6$	28510
NODE-ITERATOR (exec/sec)	3369	2
$\text{AYZ}_{3/2}$ (exec/sec)	3265	0
APPROX (exec/sec)	125	70
T	0.166	0.012
C	0.785	0.458
C^0 ($c \equiv 0$ for $d \leq 1$)	0.780	0.311
C^1 ($c \equiv 1$ for $d \leq 1$)	0.786	0.632

Table 1: Experiments on Real Networks

Figure 2 clearly demonstrates the efficiency of GEN and also of APPROX. As to be expected APPROX starts with a already relatively high constant and than grows very slowly with n.

Acknowledgements

We thank Ulrik Brandes, Marco Gaertler and Christoph Gulden for fruitful discussions and for hinting to relevant work. We further thank the reviewers for their very constructive remarks.

References

[1] N. Alon, R. Yuster, and U. Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997.

[2] A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286:509–512, 1999.

[3] A. Barrat and M. Weigt. On the properties of small-world network models. *The European Physical Journal B*, 13:547–560, 2000.

[4] B. Bollobás and O. M. Riordan. Mathematical results on scale-free random graphs. In S. Bornholdt and H. G. Schuster, editors, *Handbook of Graphs and Networks: From the Genome to the Internet*, pages 1–34. Wiley-VCH, 2002.

[5] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251–280, 1990.

[6] S. Eubank, V. A. Kumar, M. V. Marathe, A. Srinivasan, and N. Wang. Structural and algorithmic aspects of massive social networks. In *Proceedings of the 14th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA'04)*, pages 718–727, 2004.

[7] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the Internet topology. In *Proceedings of SIGCOMM'99*, 1999.

[8] F. Harary and H. J. Kommel. Matrix measures for transitivity and balance. *Journal of Mathematical Sociology*, 6:199–210, 1979.

[9] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):713–721, 1963.

[10] P. Holme and B. J. Kim. Growing scale-free networks with tunable clustering. *Physical Review E*, 65(026107), 2002.

[11] X. Li, D. Leonard, and D. Loguinov. On reshaping of clustering coefficients in degree-based topology generators. In S. Leonardi, editor, *Algorithms and Models for the Web-Graph: Third International Workshop, WAW 2004, Rome, Italy, October 16, 2004, Proceeedings*, volume 3243 of *Lecture Notes in Computer Science*, pages 68–79. Springer-Verlag, 2004.

[12] M. E. J. Newman, D. J. Watts, and S. H. Strogatz. Random graph models of social networks. *Proceedings of the National Academy of Science of the United States of America*, 99:2566–2572, 2002.

[13] D. J. Watts and S. H. Strogatz. Collective dynamics of "small-world" networks. *Nature*, 393:440–442, 1998.