Bachelor Thesis

Utrecht University

Faculty of Science

Department of Mathematics

Algorithms for Determining the Clustering Coefficient in Large Graphs

Author:
Coen Boot

Supervisor:
Prof. Dr. R.H. Bisseling

January 25, 2016

1 Introduction

Networks are everywhere around us. Obvious examples are road-maps or social networks, such as Facebook and Twitter. However, there are large numbers of less visible networks, like neural networks or food chains. Because of the growing interest in understanding these networks, much analysis is done on networks, using different quantitative measures for specific properties, called metrics.

There exist many metrics, which do provide together much information about a network. Very basic metrics are the number of vertices or edges, which give an indication of the size of the graph. When a network has a high edges/vertices ratio, the network is more interconnected than a network with a lower ratio. The metric of connectivity shows whether there exists a path between every pair of vertices in the graph. Other metrics that can be thought of are the length of the shortest cycle and the diameter (length of the longest shortest path between all pairs of vertices) of graph.

One particular metric where this thesis will be about, is the clustering coefficient, defined by Watts and Strogatz in 1998 [1]. The clustering coefficient of a vertex v is defined by the number of unordered pairs of neighbours of v which are directly connected to each other, divided by the total number of unordered pairs of neighbours of v. The clustering coefficient of a graph is defined by the average clustering coefficient of its vertices.

When a specific vertex has a high clustering coefficient, it is located inside a clique (small completely interconnected group within the graph), in contrast to vertices with a low clustering coefficient. When a graph has a high clustering coefficient, it will contain many clusters or cliques, whereas graphs with a low clustering coefficient will not. When this is translated to concrete examples, more specific conclusion can be extracted from the clustering coefficient. When a graph represents a Facebook-network, the clustering coefficient indicates to which extent people have mutual friends. Considering a brain network, a significant low clustering coefficient could be a sign of spinal cord injuries or diseases like schizophrenia, autism and Alzheimers disease [8] [9].

Although computers are becoming faster, the networks that are interesting for analysis become larger at a high rate, causing a growing demand for fast algorithms. An algorithm with quadratic running time will suffice for networks with less than a million vertices, but for larger networks, algorithms need to be at least faster than $\mathcal{O}(n^2)$, linear or even sub-linear. Therefore, the main question of this thesis is if such algorithms exist and how they work.

In this thesis, a short overview of the existing literature will be given in Chapter 2, followed by some basic definitions and properties in Chapter 3. The description and examination of the most significant existing algorithms for determining or estimating the clustering coefficient is given in Chapter 4. The process of implementing the selected algorithms and looking for further optimization will be described in Chapter 5. In Chapter 6, some tests and their results are described, after which some conclusions will follow in Chapter 7.

2 Existing Literature

The problem of calculating the clustering coefficient in an efficient way has been studied before, from both theoretical and practical viewpoints. This chapter gives a concise summary of the work done before.

2.1 Foundation of the Clustering Coefficient

The first literature about the clustering coefficient dates back from 1998 and is written by Watts and Strogatz. In this paper, the clustering coefficient is defined in order to "quantify the structural properties" of networks. The big advantage of this newly introduced metric is the fact that it is able to express for all networks to which extent it contains clusters, in contrast to already existing metrics [1].

Four years later, Watts and Strogatz published together with Newman another paper, in which they define a 'global' clustering coefficient, which will be referred to as 'transitivity' in this thesis. Furthermore, an attempt in estimating the values of metrics, such as transitivity, is made and the use of them is explained by giving several examples [2].

2.2 Conceptual Approaches for calculating the Clustering Coefficient

Before the definition of the Clustering Coefficient in 1998, Alon, Yuster and Zwick wrote a paper in 1997, in which they came up with methods for finding and counting cycles with a given length in directed and undirected graphs. One of the methods which gets described, is a method for counting cycles of length 3, which are actually closed triangles in a graph. This method splits all vertices into two groups by looking at their degree and makes use of fast matrix multiplication to count triangles of high degree vertices. The running time of this algorithm depends on the algorithm that is used for matrix multiplication [3]. Because this algorithm counts closed triangles, it can be used for determining the clustering coefficient of a graph.

In 2008, Latapy published a paper in which he proposes several algorithms for listing all triangles in a graph, or all triangles containing a specified vertex or edge [4]. Although this is a different problem than counting triangles or calculating the clustering coefficient, many techniques can easily be used for calculating the clustering coefficient.

Whereas all papers mentioned above do not tend to focus on parallelization, a recent paper by Azad, Buluç and Gilbert does. Their paper, which has been published in 2015, describes an algorithm, in both sequential and parallel form, which counts and enumerate wedges and closed triangles in graphs, based on previous work by Cohen. Both algorithms make use of matrix algebra [5]. However, the algorithms are not directly suitable for calculating the clustering coefficient, because the number of wedges or the number of triangles (i, j, k) gets calculated for a pair of vertices (i, k), instead of a given individual vertex j.

2.3 Practical Experiments about calculating the Clustering Coefficient

Schank and Wagner have written two papers, which have both been published in 2005, containing reports of several experiments. The first paper [6] compares an approximating algorithm for determining the clustering coefficient, which runs in $\mathcal{O}(1)$, with an implementation of the algorithm proposed by Alon, Yuster and Zwick. Moreover, the paper gives an algorithm for generating graphs with adjustable clustering coefficients.

The other paper [7] lists several algorithms for counting triangles as well as listing them, followed by the results of several experiments. One of the main conclusions of this paper is that the algorithm proposed by Alon, Yuster and Zwick, together with a matrix multiplication algorithm gives the best result in terms of running time compared to a simple node-iterator program.

3 Definitions and Properties

3.1 Basic Definitions

Graphs are often used to display networks, in order to retrieve a higher level of understanding. Every object or entity in a network is displayed as a point, called *vertex*, and the relations between them are displayed as a line, which is called an *edge*. This forms a pair $G = (V, E)$, where V is the collection of vertices and E the collection of edges of the graph. Furthermore, we define $m = |E|$ and $n = |V|$. Whenever a network is represented as a graph, the basic structures become quickly visible.

All graphs can be divided into two types: *directed* and *undirected* graphs. In directed graphs, $(a, b) \in E$ does not imply that $(b, a) \in E$. So all vertices have a notion of direction. In undirected graphs however, $(a, b) \in E \implies (b, a) \in E$, and so (a, b) and (b, a) are considered identical. Examples of undirected graphs are a graph of a Facebook-like network (when you are a friend of X, X is a friend of you) or a molecule-graph (when one atom A is connected to another atom B, atom B must be connected to atom A. Examples of directed graphs are road-maps (some roads are one way-roads, so when there is a road from A to B, it does not imply that the same road goes from B to A) or a graph of a Twitter-like network (when a user follows you, it does not mean that you follow that user). In this thesis, all graphs are considered to be undirected.

The *neighbourhood* of a vertex v is written as $N(v)$, and exists of all vertices which are connected to v, so $N(v) := \{x \in V \mid (v, x) \in E\}$. The *degree* of a vertex v is defined as $d(v) = |N(v)|$. A *triangle* is defined as a subgraph of 3 vertices, which are all connected to each other. So the number of triangles in the neighbourhood of a vertex v can be defined as $\delta(v) := |\{(u, w) \in E : u \neq w, (u, v) \in E \text{ and } (v, w) \in E\}|$. The number of triangles in a graph G can be calculated as

$$\delta(G) := \frac{1}{3} \sum_{v \in V} \delta(v),$$

since every triangle is counted three times (because the triangle (u, v, w) gets counted in $\delta(u)$, $\delta(v)$ and $\delta(w)$).

A *wedge* at vertex v is a subgraph of 3 vertices u, v and w, for which $(u, v) \in E$ and $(w, v) \in E$. The number of wedges as v can be calculated by $\tau(v) = \binom{d(v)}{2}$. The total number of wedges in a graph can be determined by

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

3.2 Clustering Coefficient & Transitivity

The *clustering coefficient* is defined by Watts and Strogatz [1] as follows: given three vertices u, v and w, where $(u, v) \in E$ and $(v, w) \in E$, the clustering coefficient gives the likeliness that $(u, w) \in E$. When the notation introduced above is used, the clustering coefficient at a vertex v with $d(v) \geq 2$ is given by $c(v) := \frac{\delta(v)}{\tau(v)}$. Whenever $d(v) < 2$, $c(v) = 0$. The clustering coefficient of a graph G is defined as

$$c(G) = \frac{1}{|V|} \sum_{v \in V} c(v).$$

This metric must not be mistaken with *transitivity*, which is also called *triangle density*. This measure is defined by Newmann, Watts and Strogatz in [2] as

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

The inventors of this measure thought it was equal to the clustering coefficient, but this is not the case [6]. Consider a complete graph consisting of 4 vertices and remove one edge. Then $c(G) = \frac{1}{4}\left(1 + 1 + \frac{2}{3} + \frac{2}{3}\right) = 0.833$, where $T(G) = \frac{6}{1+1+3+3} = 0.75$.

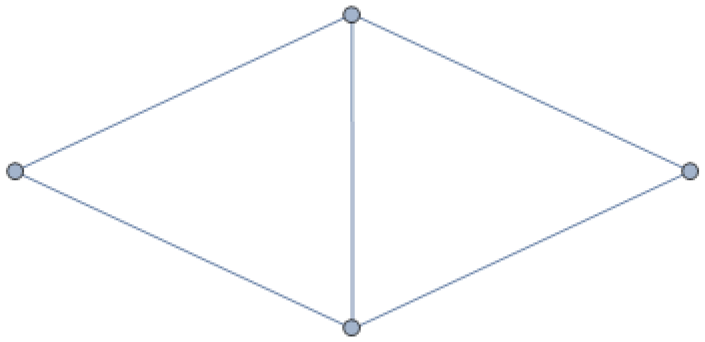

Figure 1: Counterexample showing that $c(G) \neq T(G)$.

3.3 Storing Graphs

Graphs can be stored in two ways: using an *adjacency matrix* or an *adjacency list*. An adjacency matrix A is a $n \times n$-matrix, where

$$A(i, j) = \begin{cases} 1 & \text{if an edge exists between } i \text{ and } j \\ 0 & \text{otherwise} \end{cases}$$

An adjacency list however, is a list L of length n, where $L[i]$ contains the list of neighbours of the ith vertex. Both ways of storing the edges have advantages as well as disadvantages.

4 Algorithms

There are many ways to calculate the clustering coefficient. In this chapter, different algorithms will be described and examined on several criteria, especially on time and space complexity. For all algorithms in this chapter, the assumption is made that the degree of all vertices is known.

4.1 Calculating Algorithms

In order to calculate the clustering coefficient of large graphs, two approaches are possible. The first approach is to go over the graph and calculate the actual clustering coefficient, while the other approach is to estimate the clustering coefficient in a smart way. Both approaches will be discussed in this chapter. First the focus will be on the calculating algorithms, because these give a good insight in how the clustering coefficient is calculated.

4.1.1 Brute Force - Vertices

One of the most basic algorithms is based on brute force. This algorithm performs iterations over all possible triples of vertices. For every possible triple of vertices, the existence of edges between them is checked. Whenever there are two edges between the three vertices, this triple forms a wedge, and when there are three edges, this triple forms a triangle. This algorithm has a time complexity of $\mathcal{O}\left(n^3\right)$ when an adjacency matrix is used, which causes an space complexity of $\mathcal{O}(n^2)$.

The brute force algorithm can be optimized so that pairs of edges (u, v, w) and (w, v, u) are only checked once. This can easily be done by restricting the boundaries of the loop, as follows:

```
for(int u = 0; u < n; u++)
      for(int v = u + 1; v < n; v++)
            for(int w = v + 1; w < n; w++)
                  checkEdges(u, v, w);
```

With this optimization, the running time can be reduced by a factor of 6.

4.1.2 Brute Force - Edges

Instead of looping over all triples of vertices, the algorithm can also be performed by looping over all possible triples of edges. This algorithm looks like the previous one and has a running time of $\mathcal{O}\left(m^3\right)$, which is even worse in nearly all cases. However, this algorithm works with an adjacency list and has therefore a space complexity of $\mathcal{O}(m)$, which is better in case of sparse networks.

4.1.3 Loop over adjacency of vertices

The two brute force-algorithms do not use the information whether a triple of vertices is connected or not. When the clustering coefficient of a vertex is being evaluated, it can be useful to check which other vertices are connected to the current vertex and only consider these vertices. The following algorithm uses this technique.

```
for(int u = 0; u < n; u++)
      for(int i = 0; i < d(a); i++)
            for(int j = i + 1; j < d(a); j++)
                  checkEdges(u, N(a)[i], N(a)[j]);
```

The time needed to run this algorithm is determined by $\mathcal{O}(n \cdot d'^2)$, where d' is the average number of edges connected to a vertex. Note that this is a first-order approximation, as high-degree vertices should have more influence. When this is simplified, the running time can be written as $\mathcal{O}\left(n \cdot \left(\frac{m}{n}\right)^2\right) = \mathcal{O}\left(\frac{m^2}{n}\right)$. However, in order to evaluate `checkEdges` in constant time, an adjacency matrix is needed, which leads to a space complexity of $\mathcal{O}(n^2)$.

In order to reduce the space complexity, the neighbourhoods of all vertices can be sorted and the algorithm beneath can be used together with the sorted adjacency list. Note that sorting of the neighbourhoods costs $\mathcal{O}(n \cdot d' \log d') = \mathcal{O}(m \log d') < \mathcal{O}\left(\frac{m^2}{n}\right)$, so the total time complexity stays $\mathcal{O}\left(\frac{m^2}{n}\right)$, whereas the space complexity is reduced to $\mathcal{O}(m)$.

```
for (int v = 0; v < n; v++)
      for (int i = 0; i < d(v); i++) {
            u = N(v)[i]; // Get a u connected to v
            int iu = 0, iv = i;
            while ((iu < d(u)) && (iv < d(v))) {
                  if (N(u)[iu] < N(v)[iv])
                        iu++;
                  else if (N(u)[iu] > N(v)[iv])
                        iv++;
                  else { // Neighbour in common
                        Triangles++;
                        iu++;
                        iv++;
                  }
            }
      }
```

4.1.4 Using Matrix Multiplication

Instead of simply constructing a loop over edges or vertices, the clustering coefficient can also be determined using matrix multiplication. The matrix A^3 (where A is the adjacency matrix) gives the number of paths of length 3 from i to j in the (i, j)-entry. Therefore, the diagonal entry (k, k) gives the number of closed paths of length 3 at point k. However, those paths are considered directed, so the cycle $k \to a \to b \to k$ gets counted, as well as $k \to b \to a \to k$. Therefore, the (k, k)-entry gives twice the number of closed cycles at point k.

The time complexity of this algorithm is the same as the complexity of matrix multiplication, namely $\mathcal{O}(n^3)$, and the space complexity is $\mathcal{O}(n^2)$. The time complexity could be optimized by using a fast matrix multiplication algorithm, like the Strassen algorithm, or by using sparse matrix multiplication instead of dense matrix multiplication.

4.1.5 Using the AYZ-algorithm

Besides simply multiplying the adjacency matrix, there are also more advanced techniques for finding cycles in graphs. One of the best techniques for counting triangles is proposed by Alon, Yuster and Zwick [3], whose algorithm has a theoretical running time of $\mathcal{O}(m^{1.41})$.

The basic idea behind this algorithm is that all vertices are divided into two groups: vertices with a high degree and vertices with a low degree. Let $\Delta = m^{\frac{\omega-1}{\omega+1}}$ be the separating degree between high degree and low degree vertices, where ω is the multiplication coefficient of the chosen (fast) matrix multiplication algorithm. When all vertices are pre-sorted by degree, the first step is to handle all vertices with a low degree.

For each vertex v with a low degree, the algorithm checks if two adjacent vertices u and w are connected, which costs $\mathcal{O}(1)$ using an adjacency matrix or a sorted adjacency list. The total complexity of this step is $\mathcal{O}(\Delta \cdot d')$ per low-degree vertex v, because v has at most Δ neighbours, which all have approximately d' adjacent vertices. Note again that this is a first-order approximation. Therefore, all vertices with low degree are handled within $\mathcal{O}(n \cdot d' \cdot \Delta) = \mathcal{O}(m \cdot \Delta)$.

The only triangles that are not counted, are the triangles that consist of three high degree vertices. When a new adjacency matrix A' is build, consisting only of all vertices with a high degree, the fact if such triangles do occur can be checked using fast matrix multiplication. When A'^3 gets calculated, one can easily read the number of cycles of length three by simply looking at the diagonal of the resulting matrix. The number of vertices with a high degree is at most $\frac{2m}{\Delta}$, so the matrix multiplication requires $\mathcal{O}\left(\left(\frac{m}{\Delta}\right)^{\omega}\right)$ time.

Therefore, the complexity of the total algorithm is $\mathcal{O}\left(m \cdot \Delta + \left(\frac{m}{\Delta}\right)^{\omega}\right) = \mathcal{O}(m^{1.41})$. However, the theoretically fastest matrix multiplication algorithm known is only fast when the matrices are enormous, because of the huge constant in the complexity bound, and is because of that reason not practical [10]. Therefore, this algorithm has a running time of $\mathcal{O}(m^{1.48})$ and a space complexity of $\mathcal{O}\left(\left(\frac{2m}{\Delta}\right)^2\right) = \mathcal{O}(m^{1.050})$ when the Strassen algorithm is applied [3].

4.2 Estimating Algorithms

There exist several algorithms which will give an approximation of the clustering coefficient in relatively short times. This is mostly done by taking a sample from the complete graph and calculate the local clustering coefficient of the vertices in this graph. The average of these local clustering coefficients gives an approximation of the clustering coefficient. Although these algorithms are only approximating the clustering coefficient, they do have significantly lower running times (linear, sub-linear or constant) and often produce accurate results.

4.2.1 Estimating in Constant Time

The algorithm for estimating the clustering coefficient by Schank and Wagner is described in [6] as follows:

```
int l = 0;
for(int i = 1; i < k; i++) {
      j = GetRandom(k);
      u = GetRandomVertex(N(j));
      v = GetRandomVertex(N(j));
      while (u == v)
            v = GetRandomVertex(N(j));
      if(checkEdge(u, v))
            l++;
      }
return l/k;
```

This algorithm runs in constant time, given the fact that all methods inside the loop can be evaluated in constant time. While the loop is restricted by a given number k, the total algorithm has a complexity of $\mathcal{O}(k)$, but because k can be chosen by the user, the algorithm has strictly speaking a complexity of $\mathcal{O}(1)$. However, an adjacency matrix is needed for evaluating `checkEdge` in constant time, which leads to a space complexity of $\mathcal{O}(n^2)$.

The accomplished precision of this algorithm is that a result differs less than ϵ from the true value of the clustering coefficient with chance of $(p-1)/p$ by choosing $k = \lceil \left(\ln(2v)/(2\epsilon^2)\right) \rceil$, where $p \in (1, \infty)$ [6].

4.3 Generating Graphs

In order to get data for testing the algorithms in an efficient way, Schank and Wagner wrote an algorithm for generating graphs, which is quite simple, runs in linear time with respect to $m + n$ and has the property that the clustering coefficient of the output is adjustable [6]. This generator is used for testing the implemented algorithms and is defined in the following way:

```
AddVertex(0);
AddVertex(1);
AddEdge(0, 1);

for(int i = 3; i < n; i++) {
      v = AddVertex(i);

      for(int j = 0; j < min(i - 1, d); j++) {
            u = GetRandomVertex();
            while (u == v)
                  u = GetRandomVertex();
            AddEdge(u, v);
      }

      for(int j = 0; j < o; j++) {
            u = GetRandomVertex(N(v));
            w = GetRandomVertex(N(v));
            while (u == v)
                  w = GetRandomVertex(N(v));
            if(NotExistsEdge(u, w))
                  AddEdge(u, w);
      }
}
```

Note that the number of vertices of the generated graph can be determined by the parameter n. The clustering coefficient can be adjusted by varying the parameters d and o.

5 Implementation

Not only a chosen algorithm determines the running time or memory usage of a program, but also the way the algorithm is implemented. Therefore, not only the selection of some algorithms, but also some details about the implementation will be discussed in this chapter, together with some optimizations.

5.1 Algorithms

The final implementation consists of the algorithm described in Section 4.1.3., which will be from now referred to as the 'Basic'-algorithm, together with the AYZ-algorithm as described in Section 4.1.5. Furthermore, the estimating algorithm as described in Section 4.2.1 is implemented.

The choice for these algorithms is based on the fact that they are not hard to understand and implement. Moreover, the time and space complexity is considerably low compared to other algorithms and they provide some possibilities for further optimizations. When both the Basic-algorithm and the AYZ-algorithm are combined into one program which chooses the most suitable algorithm based on the vertices/edges-ratio, good results can be achieved for both dense and sparse networks.

The code of Latapy [12], which belongs to his paper [4], was used as an example for the implementation.

5.2 Data-Structures

In order to calculate the clustering coefficient, at least all the edges need to be stored. However, the edges can be stored using an adjacency matrix or an adjacency list. Because an adjacency matrix needs one byte per entry, it will cost n^2 bytes of memory [11], assuming that the adjacency matrix consists of booleans. An adjacency list however, suffices with $2m \cdot 4$ bytes of memory [11], assuming there are less than 4.2 billion vertices and all edges are stored twice (the edge (a, b) is stored under vertex a as a link to b and under vertex b as a link to a), to keep checking the neighbours of a given vertex fast. Therefore, it is more space efficient to store all edges using an adjacency list, because in most practice-based networks, $8m < n^2$.

To make iterating over edges easy, it is also helpful to store m, n and a list of degrees of the vertices. Storing this next to the edges only cost $8n + 2 \cdot 4$ bytes of memory [11], and makes the algorithms less complicated.

5.3 Considerations and Further Optimizations

Although the time complexity of the chosen algorithm influences the running time the most, some optimizations can be done to improve the running time. Some optimizations will bring down the total running time with a constant factor, whereas other optimizations reduce the number of steps needed.

5.3.1 Multi-Threading

Instead of letting one CPU-core do all the calculations, the work can be divided over multiple cores. However, when all vertices are sorted and then divided into four equal parts, the threads will not always work simultaneously, because the work of calculating the clustering coefficient is determined by $\mathcal{O}(d \cdot d')$, where d is the degree of the given vertex. Therefore, an optimal division needs to be found to retrieve the most advantage from multi-threading.

5.3.2 Skipping Vertices with $d(v) < 2$

In most algorithms, all vertices are taken into consideration while calculating the (local) clustering coefficient. However, the clustering coefficient of vertices with $d(v) < 2$ is 0 by definition. Because the vertices are sorted by degree, it is easy to save some steps by simply skipping the vertices with a degree smaller than 2.

5.3.3 Sparse Matrix Multiplication

Instead of using dense matrix multiplication, it could in some cases be better to use sparse matrix multiplication. When the majority of the matrix-entries is equal to 0, many entries are unnecessary used in dense matrix multiplication. Implementing sparse matrix multiplication in those cases should be faster and more space-efficient, leading to better results.

5.3.4 Best of Both Algorithms

Looking at the time complexity of the Basic-algorithm and the AYZ-algorithm, none of the two algorithms is faster in all cases. The Basic-algorithm has a time complexity of $\mathcal{O}\left(\frac{m^2}{n}\right)$ and the AYZ has one of $\mathcal{O}(m^{1.48})$ when the Strassen-algorithm is used for fast matrix multiplication. Because m and n are known before calculating the clustering coefficient, the most optimal algorithm can be chosen beforehand.

6 Results

6.1 Used Graphs

To test the running time of the implemented algorithms, the clustering coefficient of the following graphs is calculated:

Name	m	n	$c(G)$	# of triangles	m^2/n
Facebook [13]	$88,234$	$4,039$	0.6055	$1,612,010$	$1,927,516$
600K (Generated)	$630,018$	$10,000$	0.2454	$5,122,401$	$39,692,268$
4M (Generated)	$630,018$	$100,000$	0.1431	$8,731,651$	$191,981,118$
Skitter [13]	$11,095,298$	$1,696,415$	0.2581	$28,769,868$	$72,568,114$
LiveJournal [13]	$34,681,189$	$3,997,962$	0.2843	$177,820,130$	$300,849,500$
37M (Generated)	$37,015,441$	$2,000,000$	0.0562	$15,748,878$	$685,071,436$
60M-I (Generated)	$59,973,402$	$5,000,000$	0.0277	$13,427,600$	$719,361,789$
60M-II (Generated)	$59,999,277$	$5,000,000$	0.0120	$5,023,429$	$719,982,648$
63M (Generated)	$63,282,257$	$4,000,000$	0.0423	$19,592,680$	$1,001,161,013$
84M (Generated)	$89,999,289$	$7,000,000$	0.0120	$7,024,089$	$1,007,982,936$
Orkut [13]	$117,185,083$	$3,072,441$	0.1666	$627,584,181$	$4,469,522,337$
383M (Generated)	$383,036,651$	$25,000,000$	0.2694	$48,646,462,257$	$5,868,683,040$

Table 1: Used graphs

All graphs are connected graphs, except for LiveJournal and Orkut, which contain $287,512$ and $6,288,363$ communities respectively [13].

6.2 Used Algorithms

The following variants of the algorithms are tested:

- Basic: Using the Basic-algorithm as described in Section 4.1.3.
- Basic-Skip: Using Basic and skipping the vertices where $d(v) < 2$ as described in Section 5.3.2.
- Basic-MT-Skip: Using Basic-Skip and applying multi-threading as described in Section 5.3.1.
- A^3: Calculating A^3 as described in Section 4.1.4, using sparse matrix multiplication.
- AYZ: Using the AYZ-algorithm as described in Section 4.1.5.
- AYZ-MT-Skip: Using AYZ with multi-threading applied as described in Section 5.3.1 on the low degree vertices and skipping the vertices where $d(v) < 2$ as described in Section 5.3.2.

- Estimation: Using the estimation algorithm as described in Section 4.2.1 with $v = 100$ and $\epsilon = 0.005$.

These algorithms were implemented in C++11 and the calculations were done on a set-up with 6GB RAM and an Intel Core i5-2450M 2.50GHz CPU with 4 cores.

6.3 Test Results

The results of the estimation algorithm are as follows:

Name	$c(G)$	Estimated $c(G)$	Difference
Facebook [13]	0.6055	0.6028	0.00278
600K (Generated)	0.2454	0.2463	0.00092
4M (Generated)	0.1431	0.1400	0.00303
Skitter [13]	0.2581	0.2576	0.00059
LiveJournal [13]	0.2843	0.2873	0.00307
37M (Generated)	0.0562	0.0562	0.00004
60M-I (Generated)	0.0277	0.0280	0.00039
60M-II (Generated)	0.0120	0.0120	0.00000
63M (Generated)	0.0423	0.0428	0.00050
84M (Generated)	0.0120	0.0123	0.00031
Orkut [13]	0.1666	0.1654	0.00123
383M (Generated)	0.2694	0.2690	0.00047

Table 2: Estimation Results

The running time of the algorithms on the given graphs are:

Name	Basic	Basic-Skip	Basic-MT -Skip	A^3	AYZ-Skip	AYZ-MT -Skip	Estimation
Facebook [13]	0.076	0.075	0.033	1.18	0.088	0.098	0.015
600K (Generated)	1.701	1.574	0.659	44.983	1.737	1.309	0.018
4M (Generated)	10.862	11.114	4.542	$> 2,000$	12.589	8.276	0.026
Skitter [13]	70.160	70.165	40.339	$> 5,400$	78.230	68.130	0.026
LiveJournal [13]	45.417	45.510	25.158	-	61.124	53.738	0.062
37M (Generated)	45.723	46.707	17.673	-	50.828	31.893	0.047
60M-I (Generated)	377.432	369.484	175.694	-	480.218	315.942	0.032
60M-II (Generated)	50.595	54.955	18.343	-	58.213	32.491	0.034
63M (Generated)	66.410	67.693	25.612	-	75.168	46.690	0.031
84M (Generated)	81.523	81.622	27.283	-	87.216	46.586	0.037
Orkut [13]	438.787	432.990	241.411	-	470.216	385.660	0.034
383M (Generated)	$22,709$	$22,624.8$	$9,828.12$	-	$> 43,000$	$> 43,000$	0.580

Table 3: Running times of the algorithms over several graphs

6.4 Analysis of the Results

6.4.1 Overall Results

Considering the running times in Table 3, it becomes clear that the Basic-Skip-MT algorithm is the fastest algorithms in all tested cases. Moreover, the Basic algorithm is the only algorithm which is able to calculate the clustering coefficient of a graph with 383 million edges in less than 7 hours.

Clearly, the A^3-algorithm performs the worst. Up to a million edges, the algorithm is capable of calculating the clustering coefficient within a reasonable timespan, but for larger graphs, the time needed becomes huge.

6.4.2 Precision of the Estimation Algorithm

The estimation algorithm with parameters $v = 100$ and $\epsilon = 0.005$ should give results that differ less than 0.005 from the real clustering coefficient with 99% chance. When the given results in Table 2 are analysed with a Single Sample One-Sided T-Test, the results satisfy this claim.

6.4.3 Running Time of the Basic Algorithm with respect to m^2/n

The time complexity of the Basic algorithm was approximated to be $\mathcal{O}\left(\frac{m^2}{n}\right)$. When the running times in Table 3 are compared to m^2/n, it becomes clear that the times are roughly proportional to the approximated time complexity.

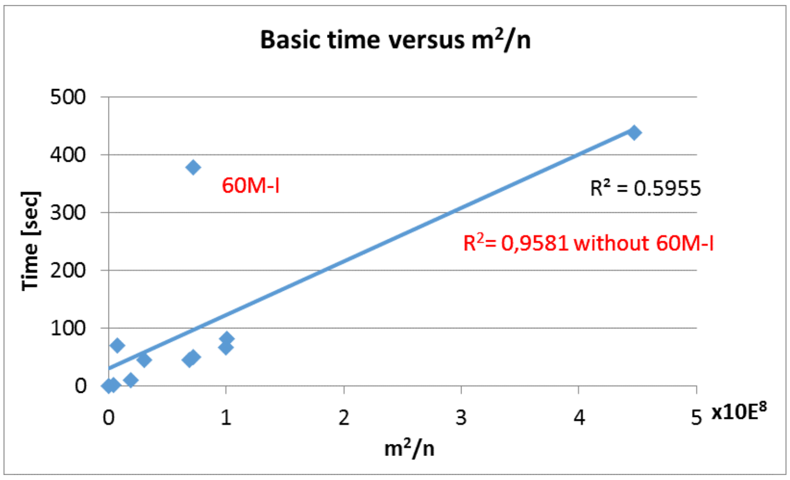

Figure 2: Graph showing the Basic times compared to m^2/n.

However, the 60M-I graph does not fit the model, while the Basic-algorithm has a substantially higher running time when calculating the clustering coefficient of this graph, as is clearly visible in Figure 2. This shows that the time complexity of the Basic-algorithm is a first-order approximation, which is not right in all cases.

6.4.4 Running Time of Basic respect to other algorithms

When the running times of the fastest algorithms (Basic-Skip, Basic-MT-Skip, AYZ-Skip and AYZ-MT-Skip) are compared to those of Basic, the times of the algorithms relate linear to the running time of Basic, as shown in Figure 3.

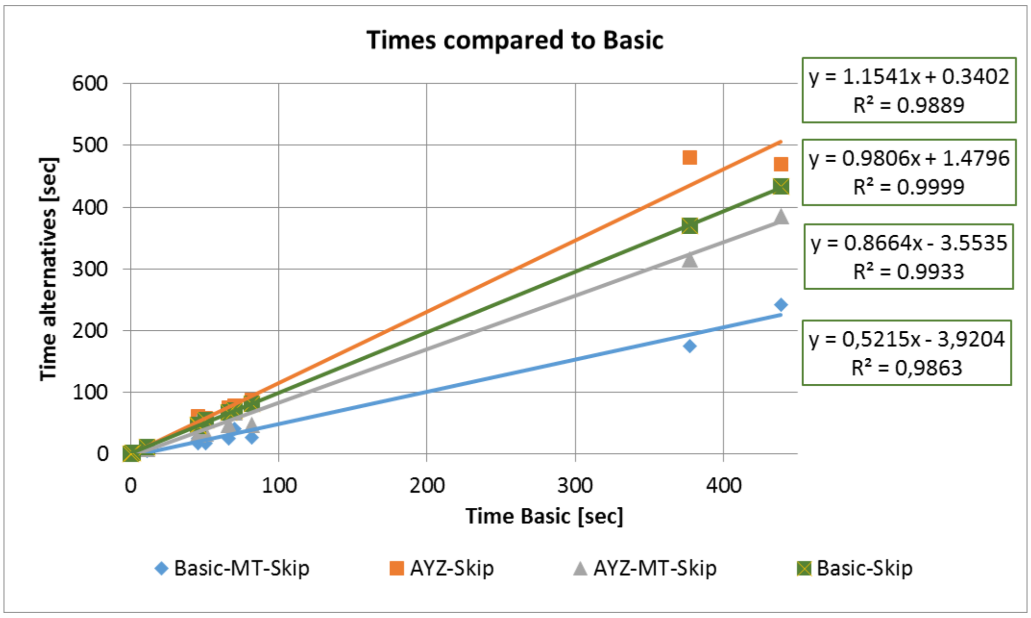

Figure 3: Graph showing times of other algorithms compared to Basic.

From the equations of the trend-lines can be extracted that the Basic-Skip algorithm is approximately equally as fast as Basic. Furthermore, the AYZ-skip algorithm is slower in all tested cases, which is also visible in Figure 3. The multi-threaded versions of both algorithms run twice as fast compared to single-threaded versions, although this should theoretically be reduced by a factor of circa 4.

Surprisingly enough, there is no direct correlation between $m^{1.48}$ and the running time of the AYZ-algorithm, although this should be the case. The reason for this could be that the Eigen-library, which is used for matrix multiplication, uses another algorithm than expected.

7 Conclusions

The clustering coefficient is a metric describing to which extent a network contains cliques or clusters, having several diverse and useful applications. Because of the growing interest in analysing networks, there is a need for fast algorithms calculating the clustering coefficient. Several techniques, like fast matrix multiplication can help calculating the clustering coefficient, but are not necessarily required.

There are different algorithms for calculating the clustering coefficient, with various time and space complexities. One of the fastest algorithms is the Basic-algorithm as described in Section 4.1.3, but in some specific cases, the AYZ-algorithm as described in Section 4.1.5 should theoretically be faster.

There exists an algorithm for estimating the clustering coefficient, which runs in constant time, and an algorithm for generating graphs with an adjustable clustering coefficient.

Several optimizations can be made to achieve faster algorithms. Though skipping vertices with $d(v) < 2$ beforehand seems a good idea, it leads in some cases to higher running times. This could be due to the fact that skipping vertices is more time intensive than simply evaluating the clustering coefficient in those vertices.

Multi-threading decreases the time needed with at least a factor of 2, but could be optimized by dividing the work over the threads as equally as possible. Algorithms involving matrix multiplication could be made more efficient by implementing sparse matrix multiplication, which saves time as well as memory in most cases.

Another way to get a fast program is by implementing multiple algorithms and deciding which algorithm should be used based on known information, like the number of edges and vertices.

The Basic-algorithm has proven to be able to calculate the clustering coefficient of graphs with over 383 million edges, but it should be able to handle even larger graphs, while only 3.1 GB of the available 6 GB RAM is used when calculating the clustering coefficient of the graph 383M.

Several test prove that in general, the running times of the Basic algorithm are proportional to m^2/n. However, when the degrees of the vertices are not uniformly distributed, the running times differ from the expected values.

Ideas for future research include finding out why the AYZ-algorithm does not correlate directly with its expected time complexity and finding techniques to optimize the multi-threading of the Basic-algorithm, to make it even faster.

References

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

[2] M. E. J. Newman, D. J. Watts, and S. H. Strogatz. *Random graph models of social networks.*
Proc. Natl. Acad. Sci. U.S.A., 99(Suppl 1):25662572, 2002.

[3] N. Alon, R. Yuster and U. Zwick. *Finding and Counting Given Length Cycles.*
Algorithms, 17:209-223, 1997.

[4] M. Latapy. *Main-memory triangle computations for very large (sparse (power-law)) graphs.*
Theor. Comput. Sci., 407:458-473, 2008.

[5] A. Azad, A. Buluç and J. Gilbert. *Parallel Triangle Counting and Enumeration using Matrix Algebra.*
Proceedings of the IPDPSW, Workshop on Graph Algorithm Building Blocks (GABB):804-811, 2015.

[6] T. Schank and D. Wagner. *Approximating Clustering Coefficient and Transitivity.*
J. Graph Algorithms Appl. vol. 9, no. 2:265275, 2005.

[7] T. Schank and D. Wagner. *Finding, Counting and Listing all Triangles in Large Graphs, An Experimental Study.*
Technical report, Universität Karlsruhe, Fakultät für Informatik, 2005.

[8] F. D. V. Fallani, J. Richiardi, M. Chavez and S. Achard. *Graph analysis of functional brain networks: practical issues in translational neuroscience.*
Phil. Trans. R. Soc. B, 369:20130521, 2014.

[9] M. E. Lynall, D.S. Bassett, R. Kerwin, P. J. McKenna, M. Kitzbichler, U. Muller and E. Bullmore. *Functional Connectivity and Brain Networks in Schizophrenia.*
J. Neurosci., 30(28): 9477-9487, 2010.

[10] C. S. Iliopoulos *Worst-Case Complexity Bounds on Algorithms for Computing the Canonical Structure of Finite Abelian Groups and the Hermite and Smith Normal Forms of an Integer Matrix.*
SIAM J. Comput., 18(4):658-669, 1989.

[11] Microsoft Developer Network. *Data Type Ranges.*
https://msdn.microsoft.com/en-us/library/s3f49ktz.aspx

[12] M. Latapy. *Triangle Computations.*
https://www-complexnetworks.lip6.fr/ latapy/Triangles/

[13] J. Leskovec and A. Krev. *SNAP Datasets: Stanford Large Network Dataset Collection.*
http://snap.stanford.edu/data, 2014.