TRIANGLE COUNTING IN LARGE NETWORKS

A PROJECT
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

YANG, CHANGYU

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

ADVISOR: JAMES, KANG

May, 2012

© YANG, CHANGYU 2012
ALL RIGHTS RESERVED

Acknowledgements

I should acknowledge the help from Professor James, Kang. She not only help me with the academic difficulties, but also care about my life. Also professor Barry James and Yongchen Qi who are my committee member.

I also like to express my gratitude to my parents who support me all the time during the time I study in America.

Finally, I'd like to thank my friends who helped me with my project.

Abstract

In network analysis, data are usually modeled as graph. Two important metrics: clustering coefficient and transitive ratio are used describe a graph. In order to calculate these two metrics in a large graph, we need to know the number of triangles in a graph. Counting the number of triangles has drawn great interest in the past few years. However, counting the triangle in large graphs such as the World Wide Web, Facebook, biology network, is very time consuming. In this project, I will introduce some counting algorithms and test their performance.

Contents

List of Tables

List of Figures

Chapter 1

Graph Concepts and Notations

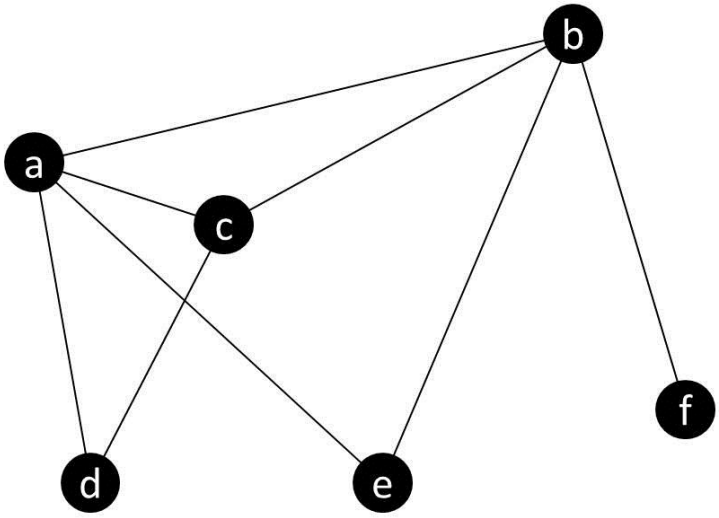

Figure 1.1: A simple graph with 6 veritices and 8 edges

A **graph G** is composed of a set of vertices $V(G)$ and a set of edges $E(G)$, denoted by $G = (V(G), E(G))$. In network analysis, G is usually considered to be an undirected and unweighted.

The **degree d(v)** of a vertex $v \in V(G)$ is the number of edges incident with v. In Fig. 1.1 $d(a) = 4, d(f) = 1$.

A **walk** is a sequence of graph vertices and graph edges such that the graph vertices and graph edges are adjacent. In Fig. 1, *a–c–d–a–b–f–b* is a walk of length 6.

A **path** is a walk with distinct edges and vertices. In Fig. 1, a–c–d–a–b–f–b is not a path because same vertices are repeated. a–c–b–f is a path.

A **neighborhood** of a vertex v in a graph G is the subgraph which consist all the vertices (neighbor of v) adjacent to v and all the edges incident with v.

A **weighted graph** is a graph with a positive number $w(e)$ assigned to each edge e. As for unweighted graphs, $w(e) = 1$ for all the edges.

A **directed graph** is a graph for which all the edges have direction. In a directed graph the edges are called arcs and each vertex v has in-degrees and out-degrees: the in-degree is the number of arcs ending with v, and the out-degree is the number of arcs starting at v.

A **complete graph** is an undirected and unweighted graph in which each pair of distinct vertices is joined by an edge. A complete graph with n vertices is denoted by K_n. So a triangle is a complete graph which can be denoted by K_3.

A **triplet** consists of three nodes that are connected by either two (open triplet) or three (closed triplet or triangle) undirected ties.

The **clustering coefficient** is a measure of the degree to which vertices in a graph tend to cluster together. The Global clustering coefficient C is defined by

$$C = \frac{3 \times numberoftriangles}{numberoftriples} \tag{1.1}$$

A **simple graph** is an unweighted, undirected graph containing no graph loops or multiple edges.

The **adjacency matrix** for a simple graph G with n vertices is an $n \times n$ matrix. Its (i, j) entry is 1 if there is an edge joining vertex v_i and v_j. The (i, j) entry is 0 if there is no such edge. The adjacency matrix is always symmetric, since the (i, j) entry should be same as the (j, i) entry. There are no self-loops (edges that start and end at the same vertex) in a simple graph, so the diagonal elements of an adjacency matrix are all equal to 0. Below is the adjacency matrix for G:

$$\begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Figure 1.2: The adjacency matrix of graph G in Figure 1.1

Chapter 2

Applications

A lot of data with interactions can be easily modeled as graphs, such as the World Wide Web, biology networks, and social networks.

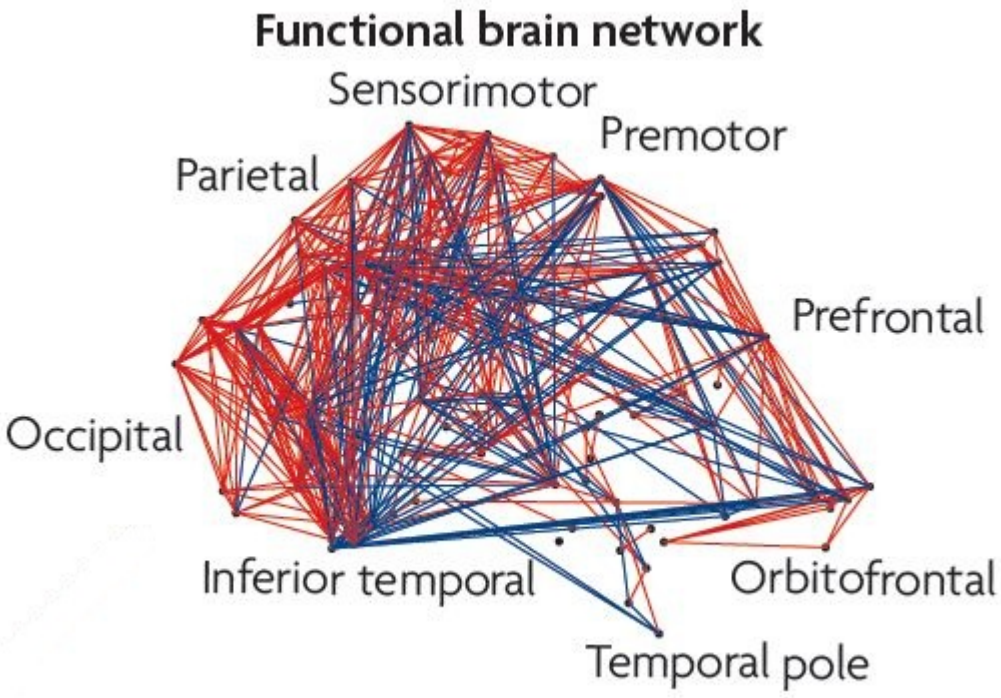

Figure 2.1: Brain functional networks [2]

Graph analysis of human brain networks is growing rapidly. Neuroscientists are interested in exploring the human brain structural and functional networks. Neuroimaging data over a wide range of scales are being investigated.The application of graph theory to the brain networks usually follow the following process: the brain networks vertices can be defined by MRI or diffusion tensor imaging data(see figure 2.1). Then the association between vertices are measured. After getting the graph, people can generate the adjacency matrix. Finally scientist can calculate the parameters of interest, and compare the difference of these metric between the patient and normal people. Graph metrics mentioned in Chapter 1 are used to detect the abnormality in clinical disorders [2]. Furthermore, counting triangles is also used to investigate protein-protein interactions [1].

Graph analysis is also widely applied in social networks. In the research of affiliation networks citeme, graph models are built to calculate the clustering coefficient. In some recent works [3],[4], triangles are used to detect network link attacks, such as spam mail and marketing phone calls.

	vertices	Edges	Description
Social Networks	75,877	405,740	Epinions network
	404,733	2,110,078	ASN
Co-authorship network	27,240	341,923	Arxiv Hep-Th
Information networks	1,222	16,714	Political blogs
	13,332	148,038	Reuters news
Web graphs	2,983,494	35,048,116	Wiki 09/25/2006
	3,148,440	37,043,458	Wiki 11/04/2006
Internet network	13,579	37,448	AS Oregon
	23,389	47,448	CAIDA AS 2004 to 2008

Table 2.1: Summary of real-word networks [1]

In real life networks,there are large number of vertices and edges included in graphs. In Table 2.1, Wikipedia has 2,983,494 vertices which are the webpages and 35,048,116 edges which are the links between the webpages by September 2006. After 40 days, the vertices increased by about 150,000 and the edges increased by nearly 2 million. So, developing a fast counting triangle algorithm is important in analyzing the large real social networks.

Chapter 3

Counting Algorithm

3.1 Exact Counting Algorithm

The real network graphs, such as the World Wide Web and Human brain network, include millions and billions of vertices. Counting the number of triangles in a huge graph is very time consuming. The nave method to count the number of triangles is to enumerate all possible $O(n^3)$ combinations of vertices and count how many of them are fully connected. This requires $O(n^3)$ time complexity.

NODEITERATOR algorithm is simpler. For each vertex in a graph, NODEITERATOR algorithm checks the neighbors of each vertex and counts how many edges exist among the neighbors. This algorithm requires $O(d_{max}^2 n)$, where d_{max} is the maximum degree in G.

EDGEITERATOR algorithm checks each edge in a graph, and counts the number of common neighbors they have. The EDGEITERATOR algorithm will be applied later for helping calculate the number of pairs of common edge triangles.

3.2 EigenTriangle Algorithm

In 2008, Tsourakakis introduced an algorithm based on calculating the eigenvalue of the adjacency matrix [5]. Tsourakakis provides two Theorems to calculate the total number of triangles in a graph (see Theorem 1) and calculate the number of triangles which include a certain vertex (see Theorem 4).

Theorem 1(EigenTriangle Theorem): The total number of triangles $\Delta(G)$ in an undirected graph is equal to the trace of A^3 and to the sum of the cubes of the eigenvalues $lambda_i$ of the adjacency matrix divided by six, i.e.:

$$\Delta(G) = \frac{1}{6} trace(A^3) = \frac{1}{6} \sum_{i=1}^{n} \lambda_i^3 \tag{3.1}$$

Before we prove theorem 1, two other theorems should be studied.

Theorem 2[6]: If A is the adjacency matrix of a graph G. Then the (i, j) entry of A^n is the number of distinct v_i–$v_j$walks of length n in G.

Proof: by induction. $n = 1$ is a trivial case. Suppose, in A^n, $a^n_{(i,j)}$ is the number of distinct v_i–v_j walks of length n in G. Since $A^{(n+1)} = A^n A$, then $a^{(n+1)}_{(i,j)} = \sum_{(k=1)}^{p} a^n_{(i,k)} a_{(k,j)}$. Every $v_i - v_j$ walk of length $n + 1$ in G must consist of a $v_i - -v_j$ walk of length n followed by the edge from v_k to v_j. Thus the theorem follows.

Theorem 3: If λ is an eigenvalue of A then λ^n is an eigenvalue of A^n

Proof: $n = 1$ is a trivial case. $Ae = \lambda e$ and e is an eigenvector corresponding to eigenvalue λ. When $n = k$, suppose if λ is an eigenvalue of A, then λ^k is an eigenvalue of A^k. So, $A^{ke} = \lambda^{ke}$. Then $A^{(k+1)}e = AA^{ke} = A\lambda^{ke} = \lambda^k Ae = \lambda^k \lambda e = \lambda^{(k+1)e}$. Thus, by mathematical induction, the theorem follows.

With the help of Theorem 2 and Theorem 3 stated above, we can prove Theorem 1(Eigentriangle Theorem) now.

Proof of Theorem 1: Based on Theorem 2, a triangle can be considered as a walk of length 3 starting and ending at same vertex v_i, so in order to find the number of triangles in graph G, we need to calculate A^3 first. The $a_{(i,j)}$ entry of A^3 equals the number of distinct walks of length 3 which start at vertex v_i and end at vertex v_i. Hence the diagonal element $a_{(i,i)}$ of matrix A^3 equals to the number of walks that starts and end at vertex v_i. Since we count the number of triangles 3 times because each triangle has 3 vertices, then the trace of A^3 is three times the total number of triangles. Furthermore, we count each triangle two times because the graph G is undirected (i.e. we count the same triangle twice as *i–j–k–i* and *i–k–j–i*). So the sum of diagonal element of A^3 equals to six times the number of triangle in G. Hence, $\Delta(G) = \frac{1}{6} trace(A^3)$. We know that $trace(A) = \sum_{(i=1)}^{n} \lambda_i$. Finally, by Theorem 3, $\Delta(G) = \frac{1}{6} trace(A^3) = \sum_{(i=1)}^{n} \lambda_i^3$

Theorem 4 (EigenTriangleLocal) The number of triangles Δ_i that node i participates in, can be computed from the cubes of the eigenvalues of the adjacency matrix.

$$\Delta_i = \frac{\sum_i \lambda_j^3 u_{(i,j)}^2}{2} \tag{3.2}$$

Proof: Since $A_{(n\times n)}$ is a symmetric matrix, so A can be diagonalized. By eigen decomposition, $A = U_n \Sigma U_n^{'}$ where Σ is a diagonal matrix with all eigenvalues on its diagonal. U_n is an orthogonal matrix with columns being the eigenvectors of A. Thus $A^3 = U_n \Sigma^3 U_n^{'}$ according to [7]. $u_{(i,j)}$ is the jth element of ith eigenvector. Furthermore, each triangle is counted twice because the graph is undirected.

3.3 Two Examples using the Eigentriangle Algorithm

3.3.1 Calculate the number of triangles in G of Fig. 1.1

From Fig.1.1, it is easy to find that there are 3 triangles in the graph G. If we apply the Eigentriangle algorithm to the adjacency matrix of graph G in Fig. 1.2, we can get the same result. The table below illustrates the calculation by Eigentriangle algorithm.

i	1	2	3	4	5	6
λ_i	3.0143257	0.848136	0.1966905	-0.7248244	-1.4779844	-1.8563434
λ_i^3	27.3886425	0.6100936	0.0076094	-0.3808013	-3.2285651	-6.3969792
	$\sum \lambda_i^3 = 18$			$\Delta(G) = \frac{1}{6}\sum \lambda_i^3 = 3$		

Table 3.1: Eigenvalue of matrix in Fig 1.2

In Fig 1.2, the dimension of the matrix is 6×6, so there are 6 eigenvalues. Take the sum of the cube of these 6 eigenvalues and we get 3, which is the number of triangles in Fig. 1. Furthermore, from this example, we can see that the small eigenvalue contributes little to the totally number of triangles. Thus, we do not need to calculate all the eigenvalues to estimate the number of triangles. The eigenvalue calculation process can stop at a reasonable level in order to reduce the counting time.

3.3.2 Calculate the Number of Triangles of a Complete Graph K_n

There is another case which can be calculated by hand. Consider the complete graph K_n with n vertices ($n \geq 2$). By definition, there exists an edge between any two vertices.

So we can calculate the number of triangles in K_n by using combinations, and the total number of triangles in K_n is $\Delta(K_n) = \binom{n}{3} = \frac{(n(n-1)(n-2))}{6}$. triangles. On the other hand, we will use the Eigentriangle Algorithm to calculate the number of triangles in K_n. The adjacency matrix of K_n is an $n \times n$ matrix with diagonal entries $a_{(i,i)}$ equal to 0 and all the other entries $a_{(i,j)}$ equal to 1. Then we need to find the eigenvalues of such a matrix. Some linear algebra tricks can make this problem very easy to solve.

- Step 1: Add all the other rows to the first row, then each entry in the first row become $n - 1$.
- Step 2: By the N—Linear Property of matrix, factor out $n - 1$ from Row 1. So the entries in the first row are all equal to 1.
- Step 3: Subtract the first row from the other rows, and then multiply the $n - 1$ back to the first row. We get an upper triangle matrix T_n with $a_{(1,1)} = n - 1$ and $a_{(i,i)} = -1$ for i from 2 to n. Thus, in triangle matrix, eigenvalues of K_n equals to the diagonal element of T_n. So $\lambda_1 = n - 1$, $\lambda_2 = \lambda_3 = ...\lambda_n = -1$.

Hence, by the Eigentriangle Algorithm, $\Delta(K_n) = \frac{1}{6}\sum_{(i=1)}^{n} \lambda_i^3 = \frac{1}{6}[(n-1)^3 + \sum_{i=2}^{n}(-1)^3] = \frac{1}{6}[(n-1)^3 - (n-1)] = \frac{1}{6}n(n-1)(n-2)$. This result is the same as what I just got by using combinations.

3.4 Trace or Eigenvalue, which is better?

Although the Eigentriangle Algorithm providse a fast way to find out the number of triangles in a graph without actual counting, the calculation is still time consuming when the graph is large. If a graph contains n vertices, we need to handle the $n \times n$ matrix which has n^2 entries. Theorem 1 provides two ways to find out the total number of triangles by either calculating the trace of the adjacency matrix or finding the sum of eigenvalues of the adjacency matrix. We need to use matrix multiplication to find the cube of the adjacency matrix A before we apply the trace method, but we do not need to do this if we apply eigenvalue method. However, both the matrix multiplication process and the eigenvalue calculation process are time consuming when the matrix is large. I used the R language for this project. I ran all my calculations on a machine with

a quad-processor Intel(R) Xeon W3530 2.80GHz with 4GB of RAM. The chart below is the system time for running the Eigentriangle Algorithm the two different ways.

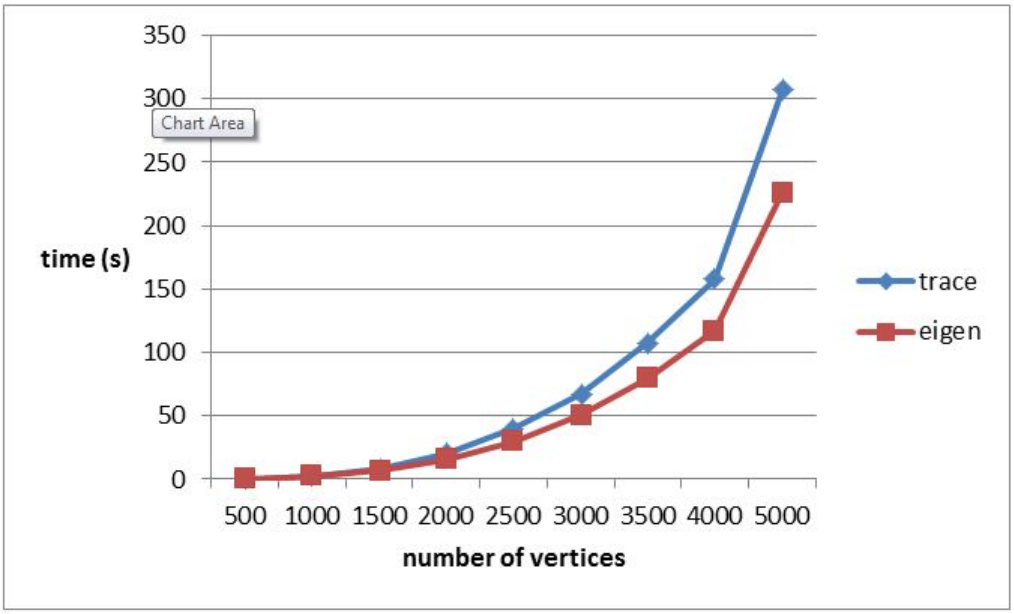

Figure 3.1: The system time for matrix multiplication and eigenvalue calculation

The chart in Fig. 3.1 indicates that finding the eigenvalue is faster than calculating the matrix multiplication. The Arnoldi—Lanczos Method is a well—studied method for calculating eigenvalues for symmetric matrix (adjacency matrices are all symmetric). Fortunately, the R language uses the ARPCACK routine to implement Arnoldi-Lanczos Method when the matrix is symmetric [R-manual]. Furthermore, there is another benefit to using eigenvalues to calculate the number of triangles. From Theorem 1, the sum of eigenvalues equals $\frac{1}{6}$ of the total number of triangles. So, if the eigenvalue λ_i is very small, it can only contribute little to the total number of triangles, especially when the graph is large. So Tsourakakis mentioned a tolerance parameter: tol in [5]. When the eigenvalue is smaller than tol, the algorithm will stop looping. This again significantly reduces the running time for Eigentriangle algorithm.

3.5 R language Simulation of Eigentriangle Algorithm

I use the R language to simulate the Eigentriangle counting process. I run the exact counting here, so I do not include the parameter *tol*. The detail code is in Appendix A.1

- Step1: Create a random graph. Since the graph can be modeled by an adjacency matrix with 0, 1 entries and 0 in the diagonal. In Fig 4, the example is based on a graph with 1000 vertices (dim=1000). Furthermore, using rbern command can create a series of Bernoulli random variables. The parameter p (I named it density in the code) of the Bernoulli variable can be used to control the number of edges in the graph, $0 \leq p \leq 1$. The larger p is, the more edges in the graph. When p=1, the adjacency matrix indicates a complete graph. When p=0, there are no edges in the graph. The MCMCpack is an R package for creating symmetric matrix, you only need to put the uppertriangular elements of the matrix and the package can automatic generate the remaining elements.

- Step 2: Use the eigen command to calculate the eigenvalue of the random symmetric matrix created in step 1, then sum the cube of these eigenvalues. Finally multiply the sum by $\frac{1}{6}$.

Chapter 4

Doulion

In order to increase the speed of triangle counting process, Tsourakakis introduce another algorithm in [1]. The new algorithm is named as Doulion. Doulion is useful when the size of the graph exceeds the available memory. Doulion can be used together with any other triangle counting algorithms to reduce the number of edges in a graph. For each edge, Doulion tosses a coin which has probability p to get a head. The edge will stay if the coin faces head and the edge will be deleted if the coin faces tail. After Doulion, the remaining edges will be reweighed.

4.1 Doulion Algorithm

The following is the Doulion algorithm:

- Step 1: Pick a coin with desired probability (or simulate this in software), and toss the coin for each edge. Edges are kept with probability p and with probability $1 - p$ edges are deleted. Then we get a new graph with fewer edges called $G^{'}$
- Step 2: Re—weight an edge with weight equal to $\frac{1}{p}$ if this edge survives after Doulion.
- Step 3: Count each triangle in $G^{'}$. Since all edges in $G^{'}$ have weight $\frac{1}{p}$, each triangle is counted as $\frac{1}{p^3}$.

4.2 Mean and Variance of Doulion

Doulion has some good properties. Since we apply probability to solve a graph theory problem, the first thing we interested in is the expected value of the number of triangles in $G^{'}$($G^{'}$ is the reduced graph after applying Doulion Algorithm). I will introduce some variables before showing the proof.

δ_i is an indicator variable for the ith triangle in $G^{'}$. So $\delta_i = 1$ if the ith triangle exist in $G^{'}$ and $\delta_i = 0$ if such triangle does not exist in $G^{'}$. X is the number of triangles in $G^{'}$ and Δ is the number of triangles in the original graph G.

Theorem 4: The expected number of triangles in $G^{'}$ is equal to the actual number of triangles in G:

$$E[X] = \Delta \tag{4.1}$$

Proof: The random variable X is the sum of the indicator variables δ_i multiplied by $\frac{1}{p^3}$. So, by the property of expected value, $E[X] = E[\sum_{i=1}^{\Delta} \frac{1}{p^3}\delta_i] = \sum_{i=1}^{\Delta} E[\frac{1}{p^3}\delta_i] = \frac{1}{p^3}\sum_{i=1}^{\Delta} E[\delta_i] = \frac{1}{p^3}\sum_{i=1}^{\Delta} p^3 = \Delta$

Theorem 5: The variance is equal to:

$$Var(X) = \frac{\Delta(p^3 - p^6) + 2k(p^5 - p^6)}{p^6} \tag{4.2}$$

where k is the number of pairs of triangles which share a common edge.

Proof: the indicator variables δ_i are identically distributed but not independent. The reason some indicator variables are not independent is that the triangles may share a common edge. In Fig. 4.1 [1], δ_i and δ_j are dependent while δ_k and δ_p are independent.

By definition, the variance of a random variable X is:

$$Var(X) = Var(\frac{1}{p^3}\sum_{i=1}^{\Delta}\sigma_i) = \frac{1}{p^6}\sum_{i=1}^{\Delta}\sum_{j=1}^{\Delta} Cov(\delta_i\delta_j) \tag{4.3}$$

In the above summation, there are Δ^2 terms in the sum. Δ of them are the variance of an indicator variable. $Cov(\delta_i\delta_j) = Var(\delta_i) = E(\delta_i^2) - [E(\delta_i)]^2 = p^3 - p^6$. Where δ_i^2 has the same indicator variable as δ_i, so $E(\delta_i^2) = p^3$. The remaining $\Delta^2 - \Delta = 2\binom{\Delta}{2}$ terms correspond to the covariances of δ_i and δ_j. Suppose k of the $\binom{\Delta}{2}$ pairs have a common edge. In that case, $Cov(\delta_i\delta_j) = E(\delta_i\delta_j) - E(\delta_i)E(\delta_j) = p^5 - p^6$. The remaining $\binom{\Delta}{2} - k$ terms have $Cov(\delta_p\delta_q) = p^6 - p^6 = 0$. Thus, we get $Var(X) = \frac{\Delta(p^3-p^6)+2k(p^5-p^6)}{p^6}$

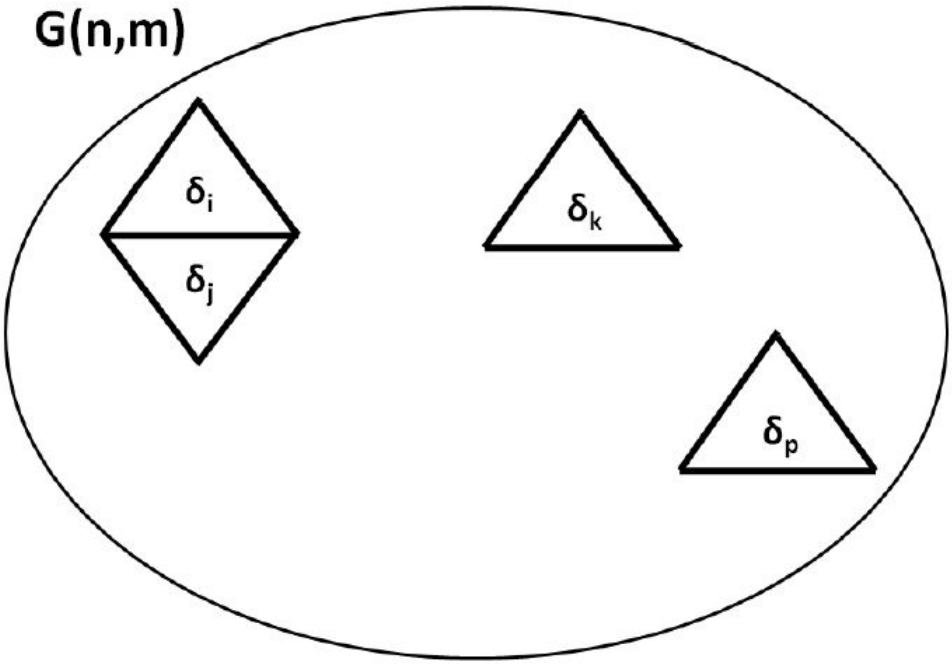

Figure 4.1: The cases should be considered when estimating the variance of Doulion. These are determined by whether the triangles are edge-disjoint or not.

Theorem 5 not only gives the factor which can affect the variance, but also indicates how to choose the Doulion probability p. Theorem 5 can be rewrite as:

$$Var(X) = \Delta(\frac{1}{p^3} - 1) + 2k(\frac{1}{p} - 1) \tag{4.4}$$

Since $p \leq 1$, then the smaller the p is, the bigger the variance. So, when we want to reduce the size of the graph by picking small p (i.e., more edge will be deleted by Doulion), we lose accuracy by making the variance large. The number of common edge triangles also contributes to the variance. The bigger k is, the larger the variance will be. From the above equation, we can see that p contributes much more to the variance than k does.

Δ^2 terms in the sum $\sum_{i=1}^{\Delta}\sum_{j=1}^{\Delta} Cov(\delta_i\delta_j)$		
Δ of the terms are the variance of indicator variables	The remaining $\binom{\Delta}{2}$ terms	
	k of pairs share a common edge	The rest of $\binom{\Delta}{2} - k$ terms are independent
$Var(\delta_i) = p^3 - p^6$	$Cov(\delta_p\delta_q) = p^5 - p^6$	$Cov(\delta_p\delta_q) = p^6 - p^6 = 0$
$\sum_{i=1}^{\Delta}\sum_{j=1}^{\Delta} Cov(\delta_i\delta_j) = \Delta(p^3 - p^6) + 2k(p^5 - p^6)$		

Table 4.1: Covariance of Δ^2 terms in the sum $\sum_{i=1}^{\Delta}\sum_{j=1}^{\Delta} Cov(\delta_i\delta_j)$

4.3 How well dose Doulion Perform?

We will bring Chebyshevs inequality here to estimate the performance of Doulion. Chebyshevs inequality can be used for any distribution. It gives a bound that most of the sample data are close to the mean. Let X be a random variable with mean and standard deviation δ. Then for any real number $k > 0$, Chebyshevs theorem is:

$$P(|X - \mu| \geq k\sigma) \leq \frac{1}{k^2}. \tag{4.5}$$

By applying Chebyshevs inequality, we get, $P(|X - \mu| \geq \varepsilon\Delta) \leq \frac{Var(X)}{\varepsilon^2\Delta^2}$. Then by Theorem 5,

$$P(|X - \mu| \geq \varepsilon\Delta) \leq \frac{p^3 - p^6}{p^6\varepsilon^2\Delta} + 2k\frac{p^5 - p^6}{p^6\varepsilon^2\Delta^2}. \tag{4.6}$$

So, the performance of Doulion depends on Δ whihch is the number of triangles in the original graph G, p which is the Doulion probability, and k which is the number of triangles which have an edge in common. The larger p, Δ and smaller k will lead to a good estimation.

4.4 R Language for the Simulation of the Doulion Algorithm

Based on all the variables defined in Appendix A.1, I wrote the R code to simulate the Doulion process in Appendix A.2

In this simulation, a very useful command which can give me the indies of the entries for which the value equals 1. These 1 entries represent the edges in the original graph G. Then I toss a coin for these entries by generating Bernoulli random variables. I keep

these 1s with probability p and with probability 1-p I change them to 0. Then I reweight each remaining 1 by multiplying the remaining 1 by $\frac{1}{p}$. After that I get a new adjacency matrix with entries equal to 0 or $\frac{1}{p}$. Finally I apply the Eigentriangle Algorithm code to find the number of triangles in the graph after applying Doulion.

We are also interested in calculating the variance of Doulion. In Theorem 5, we need to calculate the number of pairs of common edge triangles. I use an Edge Iteration Algorithm to calculate the pairs of common edge triangles in the original graph. The detail R code is in Appendix A.3.

The Edge Iteration Algorithm works great for counting the pairs of triangles which have a common edge. The algorithm I introduced here is very similar to the edge iteration method for counting triangles I mentioned in section 3.1. I check each edge in the graph, and count the number of common neighbors they have. After that, I use combination to calculate the number of pairs of triangles. This algorithm can be operated on the adjacency matrix.

- Step 1: Find the row and column index of the entries which equal 1.
- Step 2: for each entry $a_{(i,j)}$ in Step 1, check the entries in ith row and jth column, and count how many corresponding entries (i.e. $a_{(i,k)}$ and $a_{(k,j)}$ for all k from 1 to the dimension of the matrix) both equal 1. denote this by v_i for each edge.
- Step 3: for the number v_i in Step 2, calculate the combination for n choose 2 and find the sum of them.

Chapter 5

Simulation Results

I choose a random graph with 1000 vertices by generating an adjacency matrix with 1000×1000 entries. By the Eigentriangle Algorithm, I find out that the number of triangles in this graph is 56949175. By the edge iteration algorithm, I calculate that the number of pairs of common edge triangles is 41710107720. Then I simulate the Doulion process with Doulion probability p= 0.3, 0.6 and 0.9. I repeat the simulation 1000 times for each probability. Fig 5.1 is the box plot for the simulation result. The red line in Fig 5.1 is the true value of the number of triangles in the graph. It turns out that the mean of Doulion approximation is very close to the true value, and the variance is bigger when p is smaller, as expected.

By the result of simulation, we now want to test how well is Chebyshe'v bound performed.When $p = 0.3$ Chebyshevs bound indicates that there are 19.7% percent of the data for which the distance from the true value of the number of the triangles is bigger than 1000000. However, in my simulated data, only 1.2% of the data deviate that far. On the other hand, Chebyshevs bound indicates 78.8% of the data deviate from the true value by more than 50000, and there are only 25.1% of such data in my

	Variance calculated by Theorem 5	Sample Variance
p=0.3	1.96699E+11	1.776E+11
p=0.6	55820181373	54425836665
p=0.9	9290083233	8905262366

Table 5.1: Variance of the number of triangles

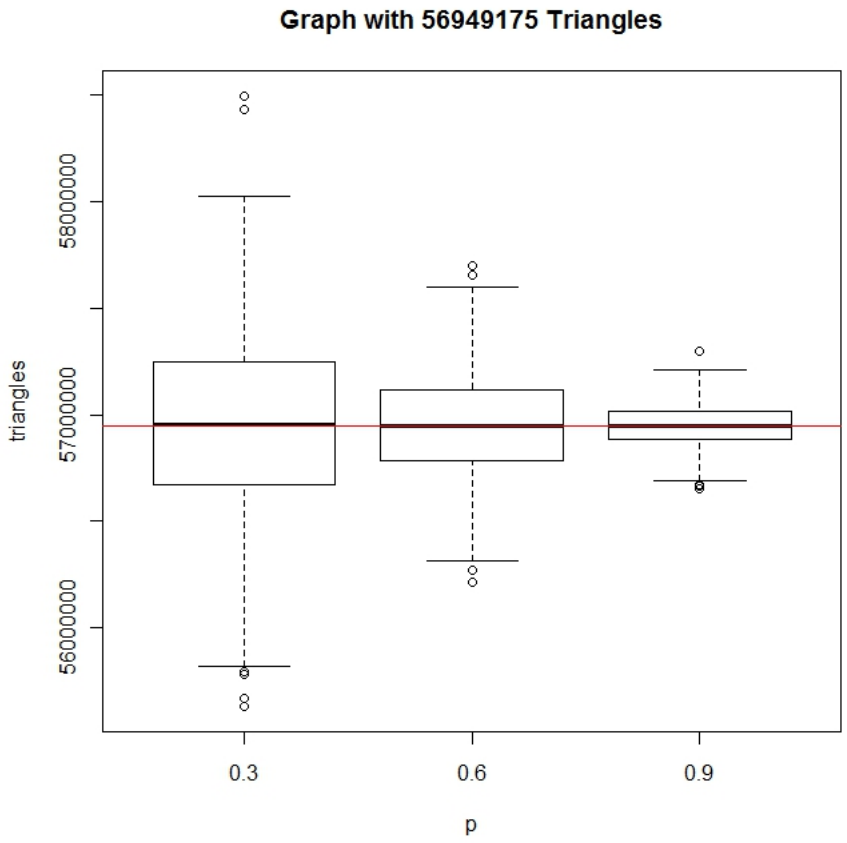

Figure 5.1: The box plot of 1000 Doulion trials with three different probability

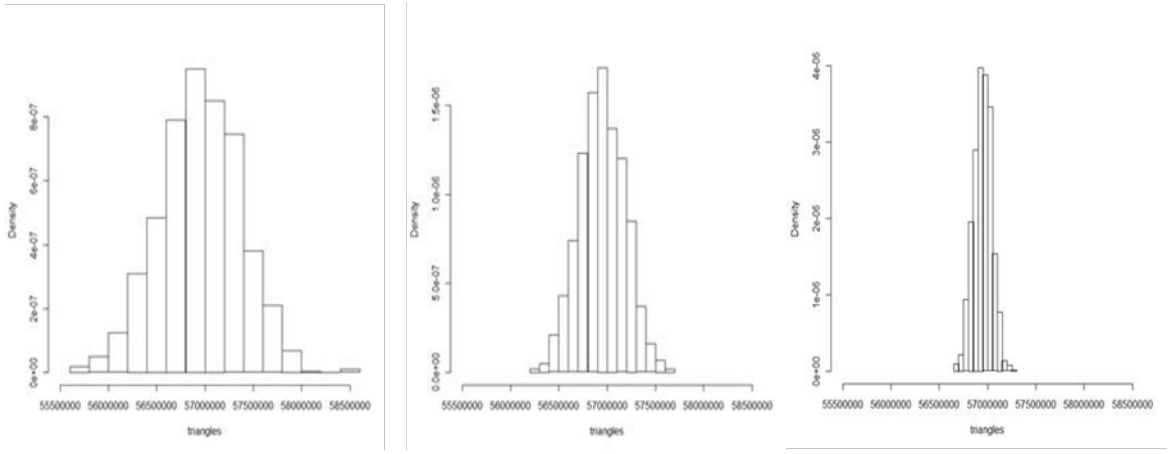

Figure 5.2: The distribution of sample data

simulation.

By Chebyshevs inequality

$P(|X - \Delta| \geq 1000000 \approx \frac{1}{57}\Delta) \leq 0.197$

$P(|X - \Delta| \geq 500000 \approx \frac{1}{114}\Delta) \leq 0.788$

From the simulation data

$P(|X - \Delta| \geq 1000000 \approx \frac{1}{57}\Delta) \leq 0.012$

$P(|X - \Delta| \geq 500000 \approx \frac{1}{114}\Delta) \leq 0.241$

Chapter 6

Conclusion and Further Study

The Eigentriangle algorithm provides a way to calculate the number of triangles in a large graph by calculating the eigenvalues of the adjacency matrix. This exact counting algorithm is faster than the traditional exact counting algorithm. By setting a tolerance level for the smallest eigenvalue, we do not need to find all the eigenvalues to do a good estimation.

Doulion can be used with any counting algorithm. When the graph exceeds the available memory, Doulion can be used to reduce the size of the graph. The accuracy of Doulion is very good when p is bigger than 0.3.

Finally, as a topic of future research, we would like to improve the bound of Doulion by a different method other than Chebyshevs bound.

References

[1] G. L. Miller C. E. Tsourakakis, U. Kang and C. Faloutsos. Doulion: counting triangles in massive graphs with a coin. *Knowledge Discovery and Dissemination*, pages 837–846, 2009.

[2] D. S. Bassett and E. T. Bullmore. Human brain networks in health and disease. *Curr Opin Neurol*, 22(), 2009.

[3] C. Castillo L. Becchetti, P. Boldi and A. Gionis. Efficient semi-streaming algorithms for local triangle counting in massive graphs. *In Proceedings of ACM KDD*, august 2008.

[4] T. Schank. *Algorithmic Aspects of Triangle-Based Network Analysis*. PhD thesis, Computer Science, University Karlsruhe, 2007.

[5] C. E. Tsourakakis. Fast counting of triangles in large real networks without counting: Algorithms and laws. *ICDM*, 0:608–617, 2008.

[6] Ronald Gould. *Graph Theory*. The Benjamin/Cummings Publishing Company, California, 1988.

[7] G. Strange. *Introduction to Linear Algebra*. SIAM, Philadephia, PA, 2003.

Appendix A

R code

A.1 Code for Simulation of Eigentriangle Algorithm

```
# install package for creating symmetric matrix
>install.packages("MCMCpack")
# install package for generating Bernoulli random variable
>install.packages("Rlab")
>library(MCMCpack)
>library(Rlab)
# dim is number of vertices, so the dimension of the adjacency matrix is dim*dim
>dim=1000
# higher the density is, the more edges in the graph. density=1 represent a
complete graph
>density=0.7
# n is the number of entries for the rbern function
>n=((dim**2-dim)/2)+dim
#generate bernoulli random variable
>entry=rbern(n, density)
#create a symmetric matrix
>sym=xpnd(entry, dim)
#change the diagonal element of the matrix to 0
>for(i in 1:dim) {sym[i,i]=0}
```

```
#calculate eigenvalues of  A^3
>e=eigen(sym)$value
#total number of triangles in G
>sum(e^3)/6
```

A.2 Code for Simulating Doulion Algorithm

```
# in the adjacency matrix, find out the indices with entry 1
>ind=which(entry==1)
#Generate Bernoulli random variables to substitute the 1 entries,
 i.e. Delete the edges with probability 1-p
>brv=rbern(length(ind),p)
#assign the remaining edges weight 1/p
>dourv=brv*(1/p)
# the adjacency matrix after Doulion
>for(i in 1:length(ind)){entry1[ind[i]]=dourv[i]}
>sym1=xpnd(entry1, dim)
# calculate the eigenvalue
>e1=eigen(adjacent1)$value
>sum(e1)/6
```

A.3 Code for Calculating number of pairs of common edged triangles

```
# find the index of the adjacent matrix equal 1(find the position of edges)
>a=which(sym==1)
# find the column index and row index of the 1 entry
>indc=(a)%%dim
>indr=floor((a/dim)+1)
>ind=cbind(indr, indc)
```

```
>pl=length(a)
# vector v is used to store the # of common edge triangle
>v=rep(0,pl)
# since I use mod, then column 0 means column 10
>for(k in which(ind[,2]==0)){ind[k,2]=10}
# this is a loop finding the number of common edge triangles
>for(j in 1:pl){
       for(i in 1:dim){
              if(sym[ind[j,1],i]==1&&sym[i,ind[j,2]]==1){v[j]=v[j]+1}
               }
  }
>print(sum(choose(v,2))/2)
}
```

Appendix B

List of frequently used symbol in this paper

Symbol	Defination
G	Undirected graph
Δ	Total number of triangles
$G^{'}$	The undirected graph after apply Doulion
X	The number of triangles in $G^{'}$
δ_i	The ith triangle in $G^{'}$
p	The Doulion probability
λ_i	The ith eigenvalues
A	The adjacency matrix
T_n	The upper triangular matrix with n vertics
K_n	The complete matrix with n vertices
$d(v_i)$	The degree of vertex v_i
C	The clustering coefficient
k	The number of pairs of commen edge triangles in G