MODELS AND MINING OF ON-LINE SOCIAL NETWORKS

by

Yanhua Tian, B.Sc. Tianjin University, 1999

A thesis
presented to Ryerson University
in partial fulfillment of the
requirements for the degree of
Master of Science
in the Program of
Applied Mathematics

Toronto, Ontario, Canada, 2011

I hereby declare that I am the sole author of this thesis or dissertation. I authorize Ryerson University to lend this thesis or dissertation to other institutions or individuals for the purpose of scholarly research.

Signature:

I further authorize Ryerson University to reproduce this thesis or dissertation by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Signature:

Abstract

Models and Mining of On-line Social Networks

Master of Science, 2011

Yanhua Tian

Applied Mathematics, Ryerson University

Power law degree distribution, the small world property, and bad spectral expansion are three of the most important properties of On-line Social Networks (OSNs). We sampled YouTube and Wikipedia to investigate OSNs. Our simulation and computational results support the conclusion that OSNs follow a power law degree distribution, have the small world property, and bad spectral expansion.

We calculated the diameters and spectral gaps of OSNs samples, and compared these to graphs generated by the GEO-P model. Our simulation results support the Logarithmic Dimension Hypothesis, which conjectures that the dimension of OSNs is $m = \lceil \log N \rceil$.

We introduced six GEO-P-type models. We ran simulations of these GEO-P-type models, and compared the simulated graphs with real OSN data. Our simulation results suggest that, except for the

GEO-P(GnpDeg) model, all our models generate graphs with power law degree distributions, the small world property, and bad spectral expansion.

Acknowledgements

First and foremost I offer my sincerest gratitude to my supervisor, Dr. Anthony Bonato, for his encouragement, guidance and support from the initial stage of this project and giving me extraordinary academic experiences through out the work.

I extend my thanks to the Department of Mathematics at Ryerson University, and especially to Mr. Stephen Kanellis, who supplied computer support. A special thanks to Dr. Peter Danziger and Dr. Dejan Delić for being part of my thesis committee.

Finally, I would like to thank my family, especially my husband Yong, for their constant support in all aspects of my life.

Contents

List of Figures

CHAPTER 1

Introduction of On-line Social Networks

With the development of computer and network technology, On-line Social Networks (OSNs) have become increasingly important in human society. An OSN is an internet-based platform to communicate, network, and share information among people. Examples of OSNs include Facebook and Twitter. Compared to traditional social networks, OSNs allow users to spread information quickly and to a large audience. Facebook, one of the most popular OSNs, has now more than 500 million active users and an average user has 130 friends (see [**17**]).

Computed-mediated communication stretches back to the late 1970s. Usenet, the oldest computer network communications systems, was created by two Duke University graduate students, Tom Truscott and Jim Ellis (see [**19, 20**]). By the late 1990s, with the boom of the Internet, a new generation of internet-mediated communication services began to flourish. Today there are over 200 major active on-line social networking websites (see, for example, [**14**]).

The studies of social networking go back to the 1960s. In 1967, Milgram and other researchers conducted several experiments examining the interconnectedness among human beings. Milgram and his research group proposed the theory that in human society every person can on average approximately reach another person in just six steps, which refers to the famous phrase "six degrees of separation" (see [**15**]). In 1998, Watts and Strogatz did further study on this topic. They defined the small world property for social networks, and introduced a random graph generation model that produces graphs with the small world property (see [**21**]).

From 2000 onwards, as they have become more and more popular, OSNs have attracted more attention from scientists. In 2003, Adamic et al. gave an early study of OSNs. They chose Club Nexus as a sample to investigate properties, such as distance between users and the clustering coefficient (see [**1**]). In 2005, in his Ph.D thesis, Liben-Nowell studied Livejournal and showed that this OSN follows the small world property (see [**13**]). In 2006, Kumar et al. studied the evolution of OSNs by studying with Flickr and Yahoo! 360 (see [**12**]). Moreover, Golder et al. (see [**11**]), Ahn et al. (see [**2**]), Mislove et al. (see [**6**]) have studied the OSNs Facebook, Myspace, Orkut, and Flickr.

1. Graph Theory

Graph theory is a natural tool to model and simulate the evolution of OSNs. To present an OSN by a graph, we represent people as vertices, and friendship between people as edges. For example, we retrieved a sample of the Wikipedia OSN (see [**18**]) to generate its friendship network; see Figure 1.1.

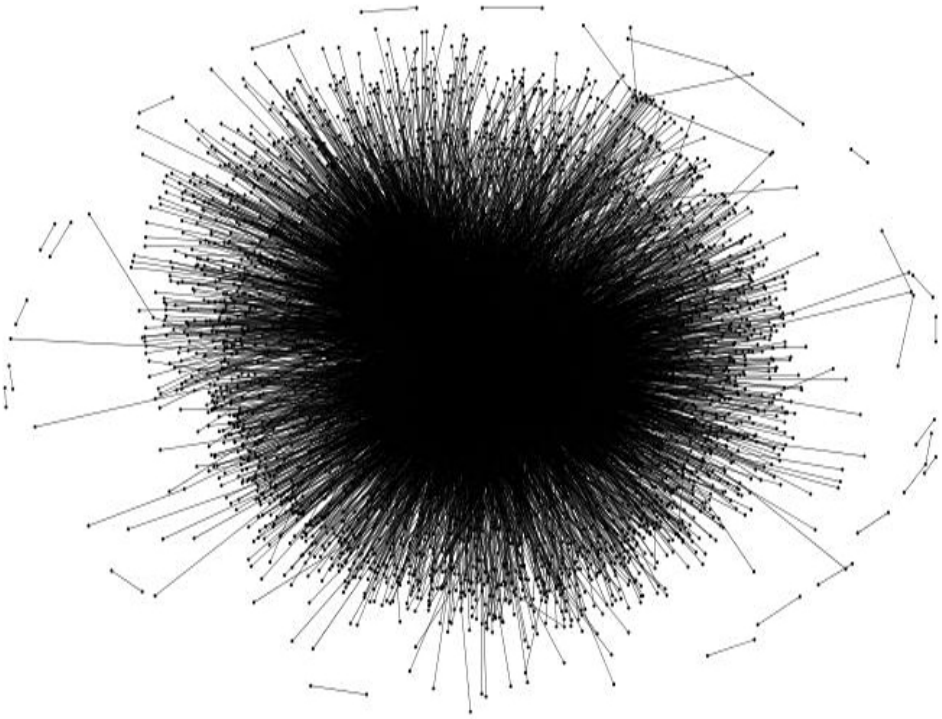


FIGURE 1.1. A friendship graph sampled from Wikipedia.

1.1. Introduction to graphs. We now give a precise definition of a graph. A *graph* G consists of a non-empty *vertex* set $V(G)$, and

an *edge set* $E(G)$ of unordered 2-elements sets from $V(G)$; we may also consider $E(G)$ as a binary relation on $V(G)$. A graph G can be written as $G = (V(G), E(G))$, or if G is clear from the context, $G = (V, E)$. In a graph G, elements of $V(G)$ are *vertices*, and the edge set $E(G)$ can be empty. We call a graph as *undirected graph* if relations between pairs of vertices are symmetric and edges are not directed. In this case, an edge is represented by uv for vertices u and v. In this thesis, we will mainly focus on *simple graphs*, which are undirected graphs without loops, and where there is at most one edge between every pair of distinct vertices.

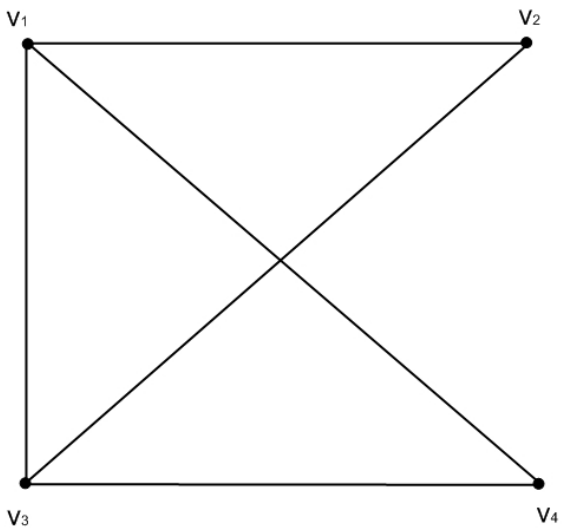


FIGURE 1.2. An undirected graph G with 4 vertices and 5 edges.

Figure 1.2 shows an example of an undirected graph. The vertex set and edge set of this graph G are:

$$\begin{aligned} V(G) &= \{v_1, v_2, v_3, v_4\}, \\ E(G) &= \{v_1v_2, v_1v_3, v_1v_4, v_2v_3, v_3v_4\}. \end{aligned}$$

The cardinality $|V(G)|$ is the *order* of the graph G. With the example of Figure 1.2, the order of the graph G is 4. We use $\deg_G(x)$ to present the *degree* of a given vertex x in graph G, which is the number of edges joined to the vertex x. With the example of Figure 1.2,

$$\deg_G(v_1) = \deg_G(v_3) = 3,$$

$$\deg_G(v_2) = \deg_G(v_4) = 2.$$

A *subgraph* of graph G is a graph whose vertex set is a subset of $V(G)$, and whose edge set is a subset of $E(G)$. A subgraph H is called an *induced subgraph* of graph G if $V(H) \subseteq V(G)$, and for any pair of vertices u and v of H, $uv \in E(H)$ if and only if $uv \in E(G)$. For example, Figure 1.3 shows a induced subgraph H of the graph G of Figure 1.2.

We now give a formal definition of random graphs. Define a probability space on graphs of a given order $N \geq 1$ as follows. Fix a

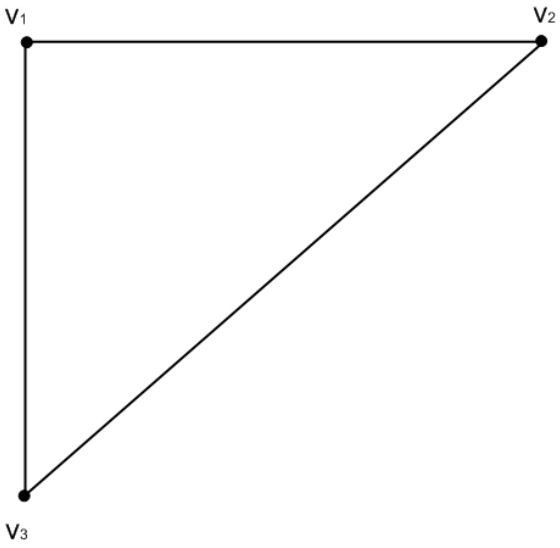


FIGURE 1.3. The subgraph H induced by $\{v_1, v_2, v_3\}$.

vertex set V consisting of N distinct elements, usually taken as $[N] = \{1, 2, \ldots, N\}$, and fix $p \in [0, 1]$. Define the space of *random graphs of order N with edge probability p*, written $G(N, p)$ with sample space equalling the set of all $2^{\binom{N}{2}}$ (labelled) graphs with vertex set V, and

$$\mathbb{P}(G) = p^{|E(G)|}(1-p)^{\binom{N}{2} - |E(G)|},$$

where $\mathbb{P}(A)$ is the probability that event A occurs.

Informally, we may view $G(N, p)$ as the space of graphs with vertex set V, so that two distinct vertices are joined independently with probability p. Even more informally: toss a (biased) coin to determine the edges of your graph. Hence, V does not change, but the number of

edges is not fixed: it varies according to a binomial distribution with expectation $\binom{N}{2}p$. Despite the fact that $G(N,p)$ is a space of graphs, we will abuse language and call it *the random graph of order N with edge probability p.*

We will consider the cases when p is fixed, and when it is a function of N. Graph parameters, such as diameter and average degree, become random variables in $G(N,p)$. We say that an event holds *asymptotically almost surely* (or *a.a.s.* for short) if it holds with probability tending to 1 as $N \to \infty$. Figure 1.4 shows a random graph with 20 vertices, and with edge drawn randomly with probability of $\frac{1}{2}$.

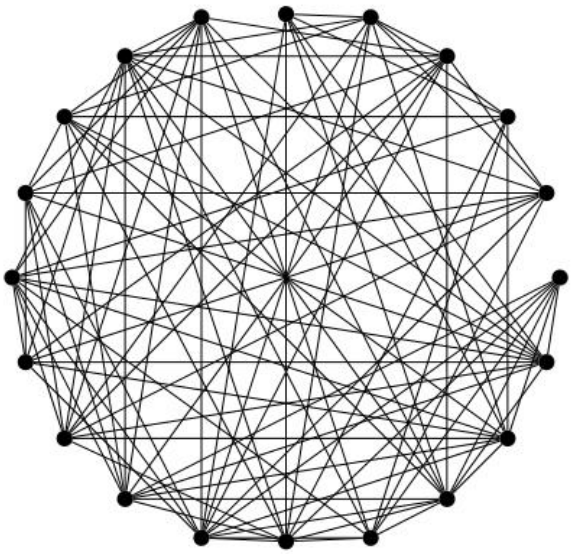


FIGURE 1.4. A random graph $G\left(20, \frac{1}{2}\right)$.

As we often use asymptotic notation to present results, we will introduce some main asymptotic notation. Let $f(x)$ and $g(x)$ be two functions defined on some subset of the real numbers, we can write: $f(x) = O(g(x))$ if

$$\lim_{x\to\infty} \sup \frac{f(x)}{|g(x)|}$$

exists and is finite. It is equivalent that there exists a constant c and a real number x_0, such that for all $x > x_0$,

$$f(x) \leq cg(x).$$

Similarly, we write: $f(x) = o(g(x))$, if

$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = 0.$$

Another asymptotic notation we introduce is

$$f(x) = \theta(g(x));$$

that is, there exists positive constants c_1 and c_2, and a real number x_0, such that for all $x > x_0$,

$$c_1|g(x)| \leq |f(x)| \leq c_2|g(x)|.$$

1.2. Power law degree distribution. Power law degree distributions are one of the most prominent properties of OSNs and other complex networks (see [**3**]). Analysis and experiments on OSNs found that the degree distribution of vertices follows a power law, which means that a large proportion of vertices have low degree, but a small (but non-negligible) proportion of vertices have substantially higher degree. In particular, the number of vertices of degree k (where k is non-negative integer) is proportional to an inverse power of k. We define the number of vertices of degree k as:

$$N_k = |\{x \in V(G) : \deg_G(x) = k\}|.$$

Then we have the formula:

$$\frac{N_k}{N} \sim k^{-\beta}, \tag{1}$$

where $\beta > 2$ is a positive real number, N is the order of graph G. If we take the logarithms on (1), then we can express the power law as:

$$\log N_k \sim \log N - \beta \log(k),$$

which gives a line with the slope of negative β.

In 2006, Kumar et al. found that the degree distribution of the Flickr OSN follows a power law (see [**12**]). Note that the power law

degree distribution may only fit a certain range of degrees; for small and large degree vertices, the log-log plot may show more noise. We analyzed Wikipedia sample data and checked the degree distribution. Figure 1.5 shows that the degree distribution log-log plot of Wikipedia follows a power law, we can see that log-log plot of out-degree of those low degree vertices follows a straight line.

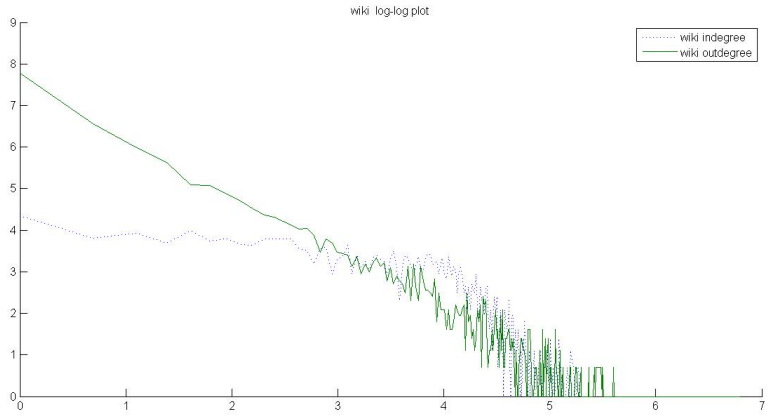


FIGURE 1.5. Wikipedia degree distribution log-log plot.

1.3. Small world property. The small world property is an important feature of OSNs, which was implicitly introduced by Milgram (see [**15**]), and explicitly by Watts and Strogatz (see [**21**]). The *neighbours* of a vertex x are defined as all vertices which are joined to x. We use notation $N(x)$ to represent the neighbour set of vertex x. We define the *distance* between pairs of vertices as the number of edges of

a shortest path joining them. We use notation $d(x, y)$ to present the distance between vertices x and y. In the example above of Figure 1.2, we have that

$$N(x) = \{y, u, v\},\ N(y) = \{x, u\},$$

$$d(x, y) = d(x, u) = d(x, v) = d(y, u) = d(u, v) = 1,$$

$$d(y, v) = 2.$$

We now give the definition of diameter, average distance, and clustering coefficient. The *diameter* of a graph is defined as the maximum distance between any two vertices (and ∞ if the graph is disconnected). We usually use the notation $\text{diam}(G)$ to represent the diameter of graph G. We define the *average distance* of an undirected graph G of order N as

$$D(G) = \frac{\frac{1}{2}\sum_{x,y \in V(G)} d(x, y)}{\binom{N}{2}}.$$

Let S be the subgraph induced by $N(x)$ and x, and let $|E(S)|$ be the number of edges in S. For a given vertex x in G with degree at least two define the *clustering coefficient* of x as

$$C(x) = \frac{|E(S)|}{\binom{\deg_S(x)}{2}}.$$

The clustering coefficient of G is the average of the clustering coefficients over all vertices of G, written $C(G)$, where

$$C(G) = \frac{1}{|V(G)|} \sum_{x \in V(G)} C(x).$$

With the graph of Figure 1.2, $\text{diam}(G) = 2$, and $D(G) = \frac{7}{12}$. Further, $C(v_1) = C(v_3) = \frac{5}{3}$, $C(v_2) = C(v_4) = 2$, and $C(G) = \frac{11}{6}$.

The small world property demands a low average distance of $O(\log \log N)$ (or a diameter of $O(\log N)$), and a higher clustering coefficient than found in a random graph $G(N, p)$ with the same order N and approximately same average degree.

1.4. Bad spectral expansion. Social networks often organize into separate clusters in which the intra-cluster links are significantly higher than the number of inter-cluster links. It is reported that social networks possess bad spectral expansion properties realized by small gaps between the first and second eigenvalues of their adjacency matrices (see [**10**]).

An *adjacency matrix* is a numerical way to present a graph. For a graph of order N, it is an $N \times N$ square matrix with binary entries. That is, entry a_{ij} can take the value of either 0 or 1, with 1 if there is an edge between the ith and jth vertices, and 0 if there is no edge

between the ith and jth vertices. With the graph of Figure 1.2, and the order of $\{v_1, v_2, v_3, v_4\}$, we can write adjacency matrix as:

$$A = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}.$$

A nonzero scalar λ and nonzero vector $\mathbf{v}$, which satisfy

$$A\mathbf{v} = \lambda\mathbf{v},$$

are called the *eigenvalues* and *eigenvectors*, respectively, of the matrix A. Note that if G is undirected, then its adjacency matrix is symmetric and so has all real eigenvalues. We sort the eigenvalues of matrix A such that

$$|\lambda_1| \geq |\lambda_2| \geq ... \geq |\lambda_{N-1}| \geq |\lambda_N|.$$

Here λ_1 and λ_2 are called the *first* and *second eigenvalues* of G. With the undirected graph of Figure 1.2, we have that $\lambda_1 = 2.56$ and $\lambda_2 = 0$.

Fix a graph G of order N with first and second eigenvalues λ_1 and λ_2, respectively. Suppose that p is chosen so that the random graph $G(N, p)$ has the same expected average degree as G, and let ρ_1 and ρ_2 be the first and second eigenvalues of $G(N, p)$, respectively. We say

that G has *bad spectral expansion* if

$$|\lambda_1 - \lambda_2| < |\rho_1 - \rho_2|.$$

2. Overview of Thesis

In this thesis, the remaining chapters focus on the following topics. In Chapter 2, we introduce the Geometric Protean models for OSNs, written GEO-P (first presented in [**7**]). We summarize the main theoretical results proved about this model in [**7**]. We simulate graphs using GEO-P, analyze their properties, and verify if graphs generated by those models follow the observed OSNs properties, such as power law degree distribution, small world property, and bad spectral expansion. In Chapter 3, we present results on real OSNs data, compute properties of graphs simulated by sampled data, and verify if the OSN samples obey the main properties described in Chapter 1. In Chapter 4, we introduce and simulate new models based on the GEO-P model. We compare these models to real OSN data. In Chapter 5, we summarize the results of the previous chapters, and state open problems.

We note that the results of this thesis are original work. Parts of Chapters 2 were included in the accepted paper [**7**].

CHAPTER 2

Geometric models for OSNs

In this chapter, we introduce the Geometric Protean (or GEO-P) model for OSNs, first presented in [**7**].

1. Introduction to the GEO-P model

The GEO-P model has vertices in a fixed metric space. We choose N vertices in a m-dimensional Euclidean space by some random process, assign vertices integers as their ranks, and determine edges by the probability based on ranks of vertices.

There are four parameters in this model: the *attachment strength* $\alpha \in (0, 1)$, the *density parameter* $\beta \in (0, 1 - \alpha)$, the *dimension* $m \in \mathbb{N}$, and the *link probability* $p \in (0, 1]$. The GEO-P model generates a sequence of graphs G_t over an infinite sequence of discrete time-steps. At each time-step t, one new vertex is born and one existing vertex dies. For each vertex x, we define a radius of influence, at each time-step we place an edge between pairs of vertices with probability p if one lies within the other's radius of influence. We begin by defining

$r(x,t)$ to indicate the (unique) rank of vertex x at time-step t. Hence,

$$r(x,t) \in [N],$$

where $[N] = \{1, 2, 3, ..., N-1, N\}$. In this model, 1 is the highest rank and N is the lowest rank. At initial time-step $t = 0$, vertices are assigned ranks by a random order. Figure 2.1 shows 20 randomly chosen vertices in the 2-dimensional unit hypercube with ranks assigned by a random order.

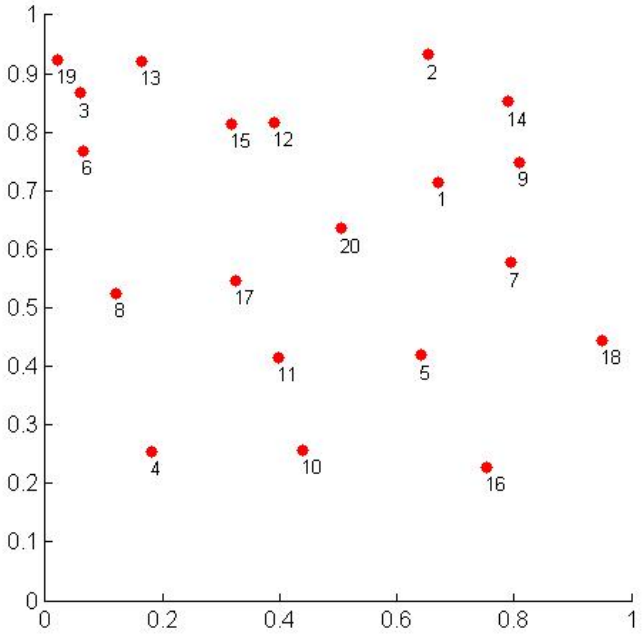


FIGURE 2.1. 20 randomly chosen vertices in a 2-dimensional unit hypercube.

At time-step t, where $t > 0$ the new vertex u is assigned a rank randomly chosen from set $[N]$, written $r(u,t)$. We name the deleted vertex as v, and its rank at time-step $t-1$ is written as $r(v,t-1)$. We assign ranks to vertices x at time-step t by

$$r(x,t) = r(x,t-1) + \delta - \gamma, \tag{2}$$

where $x \in V(G_{t-1}) \cap V(G_t)$, $\delta = 1$ if $r(x,t-1) > r(u,t)$, and 0, otherwise, and $\gamma = 1$ if $r(x,t-1) > r(v,t-1)$, and 0, otherwise. Every vertex has a unique rank at a given time-step; that is, $r(x,t) = r(y,t)$ if and only if $x = y$.

We define the *influence region* of vertex x at time-step $t \geq 0$, written $R(x,t)$, to be the ball around x with volume

$$|R(x,t)| = r(x,t)^{-\alpha} N^{-\beta}. \tag{3}$$

Since the exponent $-\alpha$ is negative, we have that the vertex with rank 1 has the largest influence region, and vertex with rank N has the smallest influence region. We assume that the volume of influence region corresponds to the ability to gain edges. Hence, higher ranked vertices are more likely to receive more edges than lower rank vertices. In Table 1 we list ranks, coordinates, and influence region volume of

20 vertices which are randomly chosen in Figure 2.1, where we choose $\alpha = 0.7$ and $\beta = 0.15$.

TABLE 1. Ranks, coordinates, and influence region volume of 20 vertices in the 2-dimensional unit hypercube.

rank	**coordinates**	**volume**
1	(0.6712 , 0.7152)	0.137
2	(0.6537 , 0.9326)	0.1488
3	(0.06 , 0.8668)	0.0878
4	(0.1807 , 0.2554)	0.1191
5	(0.6421 , 0.419)	0.2418
6	(0.0642 , 0.7673)	0.0812
7	(0.7947 , 0.5774)	0.3928
8	(0.1202 , 0.525)	0.1059
9	(0.8092 , 0.7486)	0.1634
10	(0.44 , 0.2576)	0.1273
11	(0.3989 , 0.4151)	0.0916
12	(0.3908 , 0.8161)	0.182
13	(0.1635 , 0.9211)	0.638
14	(0.7891 , 0.8523)	0.2068
15	(0.3174 , 0.8145)	0.1121
16	(0.7519 , 0.2287)	0.0958
17	(0.3258 , 0.5464)	0.1006
18	(0.9509 , 0.444)	0.0784
19	(0.0205 , 0.9237)	0.0844
20	(0.5056 , 0.6357)	0.2957

We use $\|\cdot\|_\infty$ to indicate the *infinity norm* or *maximum norm*, defined as:

$$\|\mathbf{v}\|_\infty = \max(|v_1|, |v_2|, ..., |v_{m-1}|, |v_m|),$$

where $\mathbf{v}$ is a vector in $\mathbb{R}^m$, and $\mathbf{v} = (v_1, v_2, ..., v_{m-1}, v_m) \in \mathbb{R}^m$. Let S be the unit hypercube in $\mathbb{R}^m$. We define the *torus metric* on S as

follows. For $x, y \in S$ define

$$d(x,y) = \min\{||x - y + u||_\infty \, : \, u \in \{-1,0,1\}^m\}. \tag{4}$$

The torus metric thus "wraps around" the boundaries of the unit cube, so every point in S is equivalent. The torus metric is chosen so that there are no boundary effects, and altering the metric will not significantly affect the main results. With the example of Figure 2.1,

$$d(17,11) = 0.1314, \; d(17,5) = 0.3162.$$

We define *the $m-$dimensional radius* of a vertex in terms of a parameter C_m as

$$\text{Rad}(x) \;\; = \;\; \left(\frac{R(x,t)}{C_m}\right)^{\frac{1}{m}}, \tag{5}$$

where, if m is even, then

$$C_m = \frac{\pi^{\frac{m}{2}}}{(\frac{m}{2})!},$$

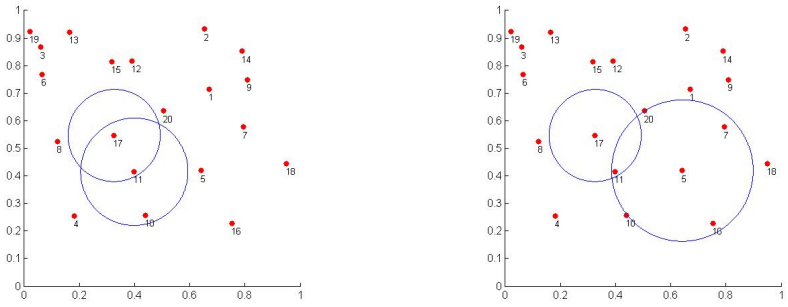
and if m is odd, then

$$C_m = \frac{2^{\frac{m+1}{2}}\pi^{\frac{m-1}{2}}}{m!!}.$$

Here, $m!!$ is defined as *double factorial* of an odd integer m; that is, for $m = 2k - 1$, and k a positive integer:

$$m!! = \prod_{i=1}^{k}(2i - 1) = \frac{(2k)!}{k!2^k}.$$

We apply (5) to calculate the influence region radius for vertices 17, 11 and 5; Table 2 shows their radius values. If we simply choose parameter $p = 1$, then for every pair of vertices there will be an edge if at least one of those two verties falls in another one's influence region. Vertex 11 falls in 17's influence region as depicted in Figure 2.2(a), so there is an edge between them; vertices 17 and 5 are too far apart to fall in each other's influence region as Figure 2.2 (b) showed. Hence, no edge is between vertices 17 and 5; see Figure 2.3.



(a) Influence region of vertices 17 and 11. (b) Influence regions of vertices 17 and 5.

FIGURE 2.2. Graph generated by 20 vertices at $p = 1$.

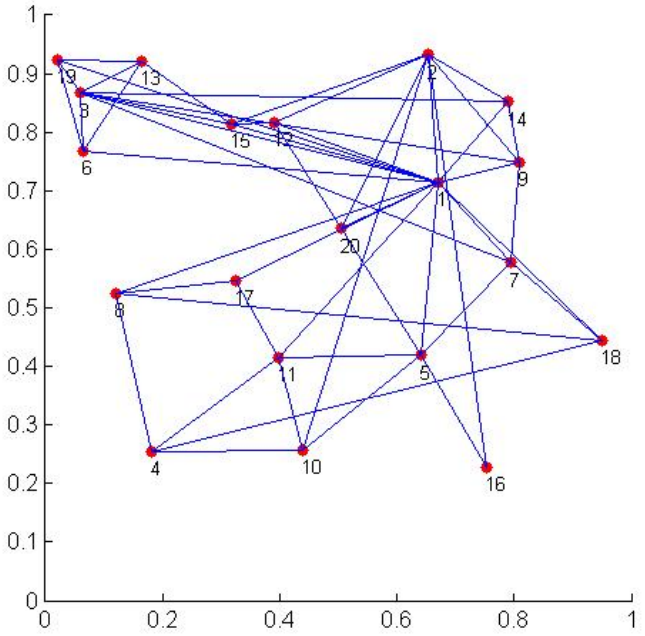


FIGURE 2.3. Graph generated by the GEO-P model, $\alpha = 0.7$, $\beta = 0.15$, and $p = 1$.

TABLE 2. Radius of vertices 17, 11, and 15.

rank	**coordinates**	**volume**	**radius**
17	(0.325834, 0.546449)	0.087807	0.1672
11	(0.398881, 0.415093)	0.119089	0.1947
5	(0.642061, 0.419048)	0.2068	0.2566

We now describe precisely how the GEO-P model generates graphs.

1. At the initial time-step $t = 0$, we randomly choose N vertices in S, and randomly assign vertex unique rank chosen from set $[N]$. An edge between pair of vertices x and y is created with probability p,

if

$$d(x, y) \leq \max\{\mathrm{Rad}(x), \mathrm{Rad}(y)\}.$$

2. At time-step $t > 0$, a randomly chosen vertex $v \in V(G_{t-1})$ dies, a new vertex u is born and is assign a rank randomly chosen from set $[N]$. We update ranks of vertices $x \in V(G_{t-1}) \cap V(G_t)$ by (2), and determine edges by influence region, distance and probability p.

It can be shown that this stochastic process converges to a stationary distribution (see [**7**]). The random graph corresponding to this distribution with given parameters α, β, m, p is called the *geo-protean* (or *GEO-P* model) graph, and is written GEO-P(α, β, m, p).

2. Results on GEO-P

We now introduce some properties of graphs generated by the GEO-P model. Recall that an event holds *asymptotically almost surely* (*a.a.s.*) if it holds with probability tending to 1 as n tends to infinity. The following theoretical results for the GEO-P model were stated and proved in [**7**].

Let $N_k = N_k(m, p, \alpha, \beta)$ denote the number of vertices of degree k in a graph of order n, generated by the GEO-P model with parameters p, α, and β. Define $N_{\geq k} = \sum_{l \geq k} N_l$ to be the number of vertices of

degree greater than or equal to k. The following theorem shows that the GEO-P model generates graphs with power law degree distribution.

Theorem 2.1 ([**7**]). *Let* $\alpha \in (0,1)$, $\beta \in (0, 1-\alpha)$, $m \in \mathbb{N}$, $p \in (0,1]$, *and*

$$n^{1-\alpha-\beta} \log^{1/2} n \leq k \leq n^{1-\alpha/2-\beta} \log^{-2\alpha-1} n.$$

Then a.a.s. *GEO-P*(α, β, m, p) *satisfies*

$$N_{\geq k} = \left(1 + O(\log^{-1/3} n)\right) \left(\frac{\alpha}{\alpha+1}\right) p^{1/\alpha} n^{(1-\beta)/\alpha} k^{-1/\alpha}. \tag{6}$$

From (6), for fixed parameters α and β, we have that

$$\frac{N_{\geq k}}{n} \sim k^{-\frac{1}{\alpha}}.$$

Hence,

$$\frac{N_k}{n} \sim k^{-\frac{1}{\alpha}-1}$$

(see, for example, [**3**]).

Theorem 2.1 demonstrates that for a range of degrees, the GEO-P model generates power law graphs with exponent

$$b = 1 + 1/\alpha. \tag{7}$$

For a graph $G = (V(G), E(G))$ of order n, define the *average degree of* G by $d = \dfrac{2|E(G)|}{n}$. The second theoretical results shows that graphs generated by the GEO-P model are dense; that is, the average degree tends to infinity with n.

Theorem 2.2 ([**7**])**.** A.a.s. *the average degree of GEO-P*(α, β, m, p) *is*

$$d = (1 + o(1)) \left(\frac{p}{1 - \alpha} \right) n^{1-\alpha-\beta}. \tag{8}$$

By the small world property, OSNs require a diameter of $O(\log n)$. Following theorem describes the diameter features of graphs generated by GEO-P model.

Theorem 2.3 ([**7**])**.** *Let* $\alpha \in (0, 1)$, $\beta \in (0, 1 - \alpha)$, $m \in \mathbb{N}$, *and* $p \in (0, 1]$. *Then* a.a.s. *the diameter of GEO-P*(α, β, m, p) *is*

$$D = O(n^{\frac{\beta}{(1-\alpha)m}} \log^{\frac{2\alpha}{(1-\alpha)m}} n). \tag{9}$$

If we assume that the dimension m grows as a logarthmic function of n, then the diameter of the GEO-P model is constant. To see this,

if we let $m = C \log n$, $C \in (0,1)$, then we have by (9) that

$$\begin{aligned}
\log D &= O\left(\frac{\beta}{(1-\alpha)C\log n}\log n + \frac{2\alpha}{(1-\alpha)C\log n}\log\log n\right) \\
&= O\left(\frac{\beta}{(1-\alpha)C\log n}\log n + o(1)\right) \\
&= O\left(\frac{\beta}{(1-\alpha)C}\right).
\end{aligned}$$

If $\xi = \dfrac{\beta}{(1-\alpha)C}$, then we have that

$$\begin{aligned}
\log D &= O(\xi) \\
&= O(1).
\end{aligned}$$

Hence,

$$D = \exp\left(O(1)\right) = O(1).$$

Recall that another main feature of small world property is a higher clustering coefficient than found in random graphs with the same expected average degree. The next theorem proves that this does indeed hold in the GEO-P model.

Theorem 2.4 ([**7**])**.** A.a.s. *the clustering coefficient of G sampled from GEO-P(α, β, m, p) satisfies the following inequality:*

$$\begin{aligned} C(G) &\geq (1+o(1))\left(\frac{3}{4}\left(1-\frac{2}{3K}\right)\right)^m \left(\frac{1-\alpha}{1+\alpha}\right) p \\ &= (1+o(1))\exp\left(O\left(\frac{m}{K}\right)\right)\left(\frac{3}{4}\right)^m \left(\frac{1-\alpha}{1+\alpha}\right) p, \end{aligned}$$

where

$$K = \left\lfloor \left(\frac{n^{1-\alpha-\beta}}{\log^3 n}\right)^{1/m} \right\rfloor - \delta,$$

with δ either 0 or 1 so K is even.

For example, when $m = o(\log n)$ it can be shown that *a.a.s.*

$$C(G) \geq \left(\frac{3}{4}\right)^{m+o(m)} = n^{o(1)}.$$

Note that for random graph $G(n,p)$, the clustering coefficient of a vertex satisfies $C(x) = p$ (see [**3**]). Hence,

$$C(G(n,p)) = \frac{1}{n}\sum_{x\in V(G)} C(x) = p.$$

Hence,

$$C(G) \geq n^{o(1)} \gg C(G(n, d/n)) = d/n.$$

That is, the clustering coefficient of graphs generated by the GEO-P model is larger than in a random graph with the same order.

Let A denote the adjacency matrix and D denote the diagonal degree matrix of a graph G. Then the *normalized Laplacian* of G is

$$\mathcal{L} = I - D^{-1/2}AD^{-1/2}.$$

Let $0 = \lambda_0 \leq \lambda_1 \leq \cdots \leq \lambda_{n-1} \leq 2$ denote the eigenvalues of $\mathcal{L}$. The *spectral gap of the normalized Laplacian* is defined as

$$\lambda = \max\{|\lambda_1 - 1|, |\lambda_{n-1} - 1|\}.$$

We state that random graphs have good spectral expansion, since $\lambda = o(1)$ (see [**8, 9**]). The next theorem shows that graphs generated by the GEO-P model have bad expansion properties, compared to random graphs.

Theorem 2.5 ([**7**])**.** *Let $\alpha \in (0,1)$, $\beta \in (0, 1-\alpha)$, $m \in \mathbb{N}$, and $p \in (0,1]$. Let $\lambda(n)$ be the spectral gap of the normalized Laplacian of GEO-P(α, β, m, p). Then* a.a.s *we have the following.*

(1) If $m = m(n) = o(\log n)$, then $\lambda(n) = 1 + o(1)$.

(2) If $m = m(n) = C \log n$ for some $C > 0$, then

$$\lambda(n) \geq 1 - \exp\left(-\frac{\alpha + \beta}{C}\right).$$

Note that in either case (1) or (2), $\lambda(n)$ is close to 1, unlike in the random graph $G(n, p)$ (where $\lambda(n)$ tends to 0).

3. Dimension of the GEO-P model

Given an OSN, its *dimension* is the minimum number m of attributes needed to identify users or a small set of users. Here attributes correspond to location, career, or hobbies, for example. We think that users with similar attributes are more likely to be friends. Hence, there should be an embedding of the OSN in m-dimensional Euclidean space where users with similar attributes are clustered together.

The results stated for the GEO-P model in the previous section point to a theoretical estimate of the dimension of an OSN. The *Logarithmic Dimension Hypothesis* (or *LDH*) states that the dimension m of an OSN satisfies

$$m = O(\log n).$$

Let the OSN order to be n, power law exponent to be b, average degree to be d, and let D be the diameter. Then (7) shows us how to estimate

for α based on the power law exponent b. Let

$$\begin{aligned} d^* &= \log d / \log n, \\ D^* &= \log D / \log n. \end{aligned}$$

Equations (8) and (9) imply that (ignoring constants)

$$d^* = 1 - \alpha - \beta, \tag{10}$$

$$D^* = \frac{\beta}{(1-\alpha)m}. \tag{11}$$

Thus, by (10) and (11) we estimate m as:

$$\begin{aligned} m &= \frac{1}{D^*}\left(1 - \left(\frac{b-1}{b-2}\right) d^*\right). \\ &= \frac{\log n}{\log D}\left(1 - o(1)\right) \end{aligned} \tag{12}$$

If D is a constant, then (12) suggests that the dimension m depends on $\log n / \log D$. That is, m grows logarithmically with n. Hence, support for the LDH comes from the GEO-P model, although it is yet to be proved in practice (see Section 3 of Chapter 3).

Table 3 predicts the dimensions of OSNs Cyworld, Flickr, Twitter, and YouTube based on (12) (see [**7**]). Let b be the power law exponent

for the in-degree distribution, n be the order of given OSN, and let m be rounded up to the nearest integer.

TABLE 3. Dimensions of the OSNs: Cyworld, Flickr, Twitter, and YouTube.

Parameter	**OSN**			
	Cyworld	Flickr	Twitter	YouTube
n	2.4×10^7	3.2×10^7	7.5×10^7	3×10^8
b	5	2.78	2.4	2.99
d^*	0.22	0.17	0.17	0.1
D^*	0.11	0.19	0.1	0.16
m	7	4	5	6

CHAPTER 3

Properties of Real OSN Data

In this chapter, we consider data sampled from real OSNs. We analyze their properties viewed as complex networks, and contrast them to graphs simulated by the GEO-P model. Our sampled data was retrieved from the OSNs Wikipedia (see [**18**]) and YouTube (see [**16**]). Their orders are 7,115 and 570,774, respectively, and there are 103,689 and 4,945,382 edges in those two samples samples, respectively. We mainly focus on investigating their degree distribution, spectral gap and diameter.

1. Properties of Real OSNs

In Chapter 1, we have shown that the Wikipedia sample follows a power law degree distribution. Figure 3.1 shows the degree distribution of a graph sampled from YouTube with order 570,774 (see [**16**]). It is evident that it follows a power law degree distribution. Figure 3.2 shows that the log-log plot is a straight line for *medium degree* vertices; that is, informally, vertices which are not of low or high degree.

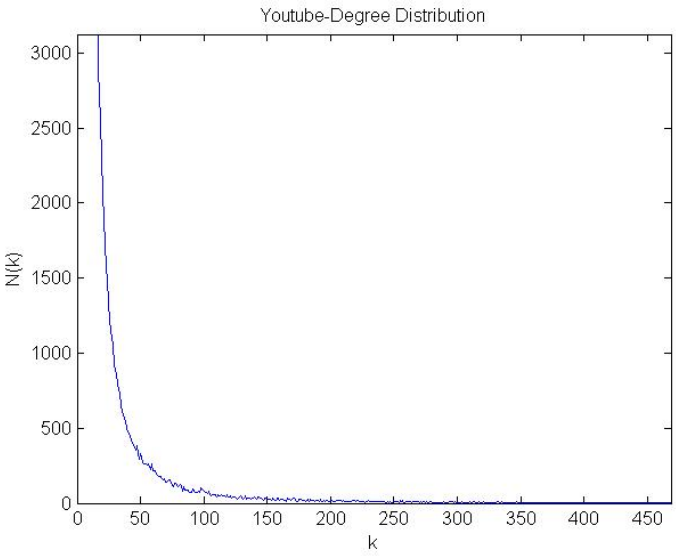


FIGURE 3.1. YouTube sample degree distribution.

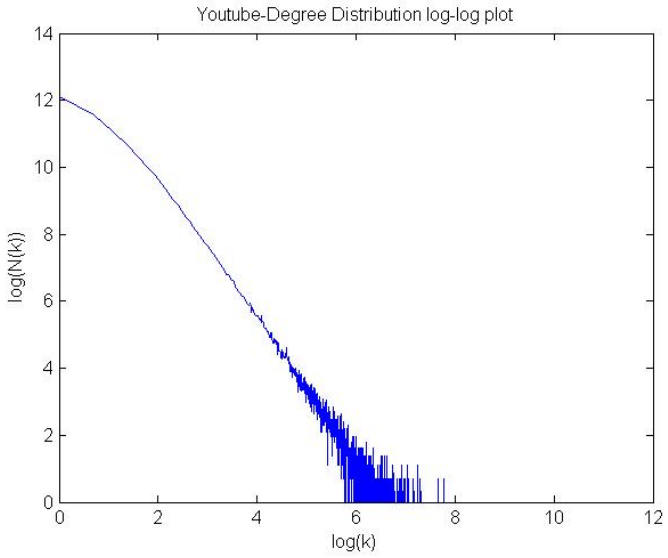


FIGURE 3.2. YouTube sample degree distribution log-log plot.

We next analyzed OSN data for their diameters. We considered graph samples from Wikipedia and YouTube. We applied the software

SNAP (see [**18**]) to calculate the diameters, and calculated that they are 10 and 23, respectively.

Finally, we checked the spectral gap of graphs in our OSNs samples. To evaluate the spectral gap, we calculate the first and second eigenvalues of their adjacency matrix. OSNs have large-scale adjacency matrices. To make the calculations more efficient, we applied the *power method* (see [**3**]) to approximate the first eigenvalue λ_1. Let A be the $N \times N$ adjacency matrix, and let $\mathbf{v_1^{(0)}}$ be any $N \times 1$ vector (here we let $\mathbf{v_1^{(0)}} = (1, 1, ..., 1)$). Then we have the iteration:

$$\begin{aligned} \mathbf{v_1^{(k+1)}} &= \frac{A\mathbf{v_1^{(k)}}}{\|A\mathbf{v_1^{(k)}}\|}, \\ \lambda_1^{(k+1)} &= \frac{A\mathbf{v_1^{(k+1)}}}{\mathbf{v_1^{(k+1)}}}. \end{aligned}$$

We run this iteration until $\left|\frac{\lambda_1^{k+1} - \lambda_1^k}{\lambda_1^{k+1}}\right| < \epsilon$, where ϵ, as the accuracy control, is a fixed positive real number.

To approximate the second eigenvalue, we apply the *deflation method* (see [**4**]). Let λ_1 and $\mathbf{v_1}$ be the largest eigenvalue and eigenvector, respectively, and let A still be the adjacency matrix, then we have that

$$A^* = A - \lambda_1 \mathbf{v_1} \mathbf{v_1^T},$$

where $\mathbf{v_1^T}$ is the transpose of $\mathbf{v_1}$.

Next, we apply power method to A^* to compute its largest eigenvalue. This in turn should be the second largest eigenvalue of the initial matrix A.

Using these eigenvalue approximation algorithms, we obtain that the spectral gap of the Wikipedia sample is about

$$|\lambda_1 - \lambda_2| = 45.1290 - 26.9872 = 18.1419.$$

We generated a random graph with the same order to contrast this result. The spectral gap of random graph of the same order $G(7115, 1/2)$ is approximately

$$|\rho_1 - \rho_2| = 3499.9 - 82.8528 = 3417.0472.$$

We now introduce the first and second neighbour set. The first neighbour set of x, written $N(x) = N_1(x)$, consists of the neighbours of x. Correspondingly, *the second neighbour set* of x, written $N_2(x)$ is the set of vertices with distance two from x.

We consider ten subgraphs S of YouTube by sampling first and second neighbour sets. More precisely, we first choose a vertex x uniformly at random from the YouTube graph. We consider the subgraph be induced by the closed first and second neighbour sets of x; that is,

the subgraph induced by

$$\{x\} \cup N_1(x) \cup N_2(x).$$

Table 1 shows the spectral gaps of our YouTube samples, and the spectral gaps of random graphs with the same order (to contrast with our OSN samples).

TABLE 1. Spectral gaps of YouTube samples and random graphs of same order.

order	**gap**	
	YouTube	**random graph**
4497	13.28	2193.76
5505	53	2696.47
5503	12.63	2680.92
6409	22.38	3115.31
5897	10.9	2923.18
3528	18.39	1727.80
7507	20.56	3619.51
6089	14.57	2997.98
5053	15.56	1709.70
6788	60.8	3223.98

With OSNs Wikipedia and YouTube as samples, the above calculation results suggest that OSNs have the following properties.

1. For some range of degree vertices (mostly, medium degree vertices), OSNs follow a power law degree distribution.
2. OSNs have a small diameter, which is one of the main features of the small world property.

3. OSNs have smaller spectral gaps than a random graph, which indicates OSNs have bad spectral expansion.

2. Properties of Graphs Generated by the GEO-P Model

We now simulate graphs using the GEO-P model in several dimensions, and compare and contrast these samples with real OSN data. Once again, we measure the degree distribution, diameter, and spectral gap.

Figures 3.3 and 3.4 show the degree distributions of graphs generated by the GEO-P model, where the order N is 7115 (the same order as our Wikipedia samples). We used dimensions m from 1 to 5, and we used two sets of values for parameters α and β. We can see that for the medium degree vertices, the log-log plot is a straight line, which suggests that graphs generated by the GEO-P model follow a power law degree distribution.

Table 2 displays the spectral gap of graphs generated by the GEO-P model. Recall that we calculated a spectral gap of a sampled random graph $G(7115, 1/2)$ to be 3417.0472 (see Section 3.1). We may conclude that graphs generated by the GEO-P model have small spectral gap. Correspondingly, they have bad spectral expansion.

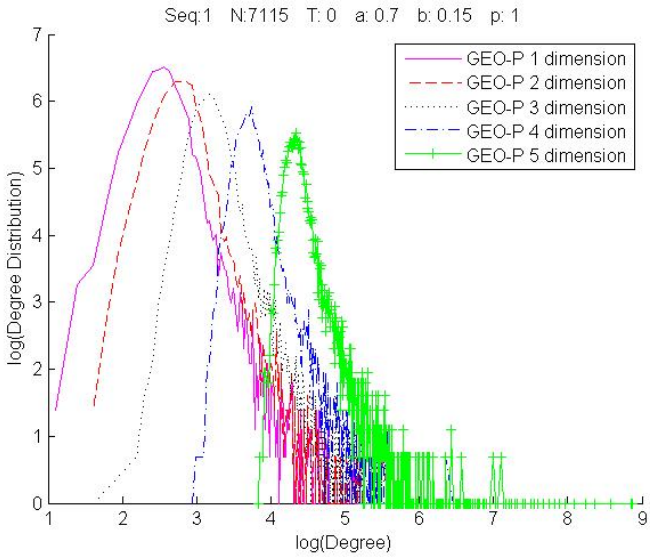


FIGURE 3.3. GEO-P degree distribution log-log plot, $N = 7115$, $\alpha = 0.7$, $\beta = 0.15$, $p = 1$.

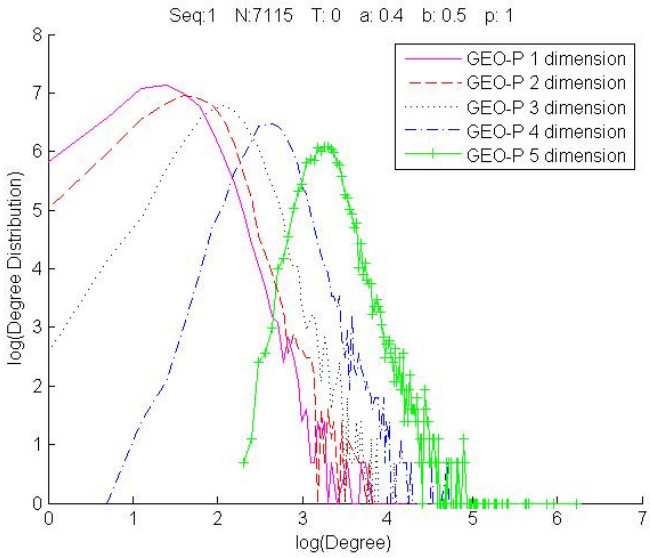


FIGURE 3.4. GEO-P degree distribution log-log plot, $N = 7115$, $\alpha = 0.4$, $\beta = 0.5$, $p = 1$.

Table 3 displays the diameters of graphs generated by the GEO-P model. We can see that graphs generated by the GEO-P model have small diameter, and the diameter decreases as m increases.

TABLE 2. Spectral gaps of graphs generated by the GEO-P model.

$N = 7115$, $p = 1$			
$\alpha = 0.7$, $\beta = 0.15$		$\alpha = 0.4$, $\beta = 0.5$	
dimension	**gap**	**dimension**	**gap**
1	23.700163	1	0.070531
2	35.04388	2	0.028725
3	57.751959	3	0.380326
4	48.840852	4	3.21985
5	95.301035	5	4.555701

TABLE 3. Diameters of graphs generated by the GEO-P model.

$N = 7115$, $p = 1$			
$\alpha = 0.7$, $\beta = 0.15$		$\alpha = 0.4$, $\beta = 0.5$	
dimension	**diameter**	**dimension**	**diameter**
1	34	1	23
2	9	2	58
3	5	3	18
4	4	4	4
5	2	5	5

3. Logarithmic Dimension Hypothesis

In Chapter 2, we introduced the Logarithmic Dimension Hypothesis (or LDH), which states the dimension m of an OSN satisfies

$$m = O(\log N),$$

where N is the order of the OSN. For our datasets, we simplify this to

$$m = \lceil \log N \rceil.$$

We now explain how we attempted to validate the LDH by comparing real OSNs with the same order graphs generated by the GEO-P model with various dimensions, and find the most appropriate dimension.

Table 1 lists the spectral gaps of 10 YouTube samples of order around 5000. We take the average of the spectral gaps, giving a gap of 24.21. On the other hand, we simulated GEO-P model to generate the same order graphs from dimension 1 to 6. Table 4 lists their spectral gaps. We can see that the best fit is between 19.2611 and 47.8424, which is for $m = 3$ and $m = 4$, respectively. Hence, we suggest that the OSNs of order 5000 have a dimension $\lceil 3.5 \rceil = 4$. It is consistent with the LDH, which demands a dimension of

$$\lceil \log 5000 \rceil = \lceil 3.69 \rceil = 4.$$

TABLE 4. Spectral gaps of graphs generated by the GEO-P model.

$N = 5000$, $\alpha = 0.7$, $\beta = 0.15$, $p = 1$			
dimension	λ_1	λ_2	**gap**
1	27.5004	27.3945	0.10593
2	60.66	54.522	6.138
3	82.9547	63.6936	19.2611
4	134.5345	86.6921	47.8424
5	221.943124	132.411872	89.531252
6	379.201601	203.734353	175.467248

Now we compare the diameters. Table 5 lists the 10 YouTube samples' diameters, we calculate the average diameter which is 7.8. Table

6 lists the diameters of graphs generated by the GEO-P model, with dimension from 1 to 6. We can see that the three-dimensional graph's diameter is close to the average diameter (with the average taken over all our samples); that is, 7.8. Hence, we have additional support for the LDH.

TABLE 5. Diameters of YouTube samples of order around 5000.

order	**diameter**	**order**	**diameter**
4497	8	3528	5
5505	9	7507	8
5503	11	6089	7
6409	9	5053	6
5897	7	6788	8

TABLE 6. Diameters of graphs generated by the GEO-P model.

$N = 5000$, $\alpha = 0.7$, $\beta = 0.15$, $p = 1$						
dimension(m)	1	2	3	4	5	6
diameter	51	10	5	4	2	2

While our results are supportive of the LDH, we do not claim to prove it. Larger samples would be necessary for that (ideally the entire network), or many more small samples. Further, there are many other graph properties we could check which is suggestive of two graphs being "similar" (although not necessarily isomorphic). We do speculate further on such similarity measures, but leave that for future work.

CHAPTER 4

New models for OSNs

We described the GEO-P model in the previous chapter. In this chapter we simulate some new models based on the GEO-P model, analyse their properties, and check if they accurately fit OSN data. These new models are similar (but not identical to) to the GEO-P model, but use different ranking schemes such as age or degree.

Recall from Chapter 2 that we use $r(x,t)$ to indicate the rank of vertex x at time-step t.

1. Ranking by Age

We first introduce the GEO-P(Age) model. In this model, an older vertex is more likely to attract more edges than a younger one. For a positive integer $t > 0$, we define

$$\text{age}(x) = t,$$

where t is the time-step when vertex x is born.

The GEO-P(Age) model has four parameters: the attachment strength $\alpha \in (0,1)$, the density parameter $\beta \in (0, 1-\alpha)$, the dimension

$m \in \mathbb{N}$, and the link probability $p \in (0, 1]$. Let S be unit hypercube in $\mathbb{R}^m$. The GEO-P(Age) model generates graphs via the following process.

1. At the initial time-step $t = 0$, we choose N vertices in S at random, randomly assign each vertex a unique rank chosen from the set $[N]$, and set all vertices ages as 0. An edge between a pair of vertices x and y is created with probability p if

$$d(x, y) \leq \max\{\text{Rad}(x), \text{Rad}(y)\}.$$

2. At time-step $t > 0$, a randomly chosen vertex $v \in V(G_{t-1})$ dies. For the remaining vertices, if $r(x, t-1) > r(v, t-1)$, then we update their ranks by

$$r(x, t) = r(x, t-1) - 1,$$

where $x \in V(G_{t-1}) \setminus \{v\}$. Hence, the rank of a vertex is determined by its age. Older vertices receive higher rank. On the other hand, a new vertex u is born and is assigned rank N. We set its age as t. Then we add edges by the method described in Item 1 above.

The GEO-P(InvAge) model is similar to the GEO-P(Age) model. The difference is we assign new vertex higher rank as follows in item (2) below.

1. At the initial time-step $t = 0$, we choose N vertices in S at random, randomly assign each vertex a unique rank chosen from the set $[N]$, and set all vertices ages as 0. An edge between a pair of vertices x and y is created with probability p if

$$d(x, y) \leq \max\{\mathrm{Rad}(x), \mathrm{Rad}(y)\}.$$

2. At time-step $t > 0$, a randomly chosen vertex $v \in V(G_{t-1})$ dies, for remaining vertices, if $r(x, t-1) > r(v, t-1)$, then we update their ranks by

$$r(x, t) = r(x, t-1) + 1,$$

where $x \in V(G_{t-1}) \setminus \{v\}$. Hence, a new vertex is more likely to be higher ranked. On the other hand, a new vertex u is born and is assigned rank 1. We set its age as t. Then we determine edges by the method described in Item (1) above.

Now we analyse degree distribution, diameter, spectral gap of graphs generated by the GEO-P(Age) and the GEO-P(InvAge) models. We applied the GEO-P(Age) and the GEO-P(InvAge) models to generated graphs embedded in $\mathbb{R}^n$, where $1 \leq n \leq 5$. Figures 4.1 and 4.2 display the degree distribution log-log plot. Tables 1 and 2 list the diameters and spectral gaps.

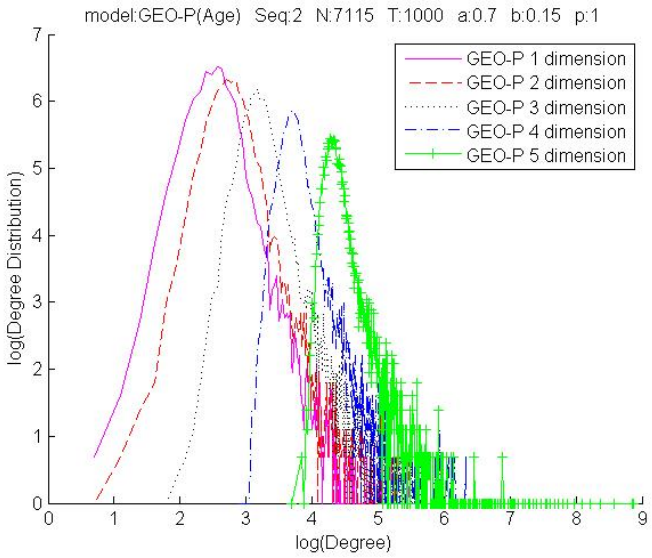


FIGURE 4.1. Degree distribution of a graph generated by the GEO-P(Age) model.

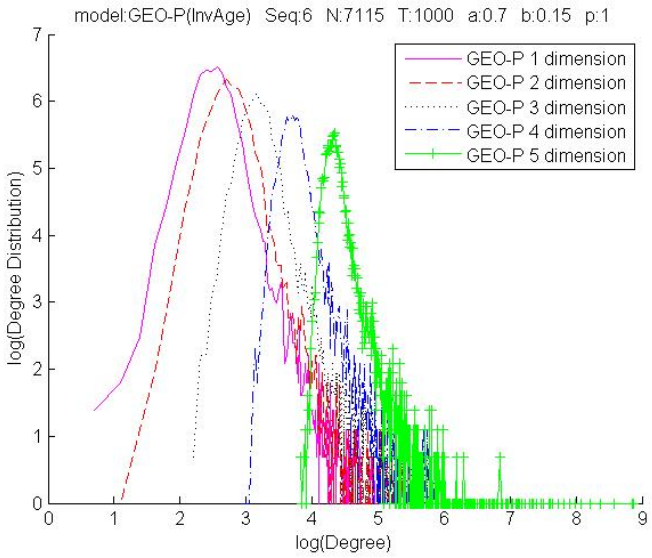


FIGURE 4.2. Degree distribution of a graph generated by the GEO-P(InvAge) model.

Comparing with the graphs generated by the GEO-P, GEO-P(Age), and GEO-P(InvAge) models, our results suggest that the models generate graphs with similar degree distributions; the diameters all decrease

TABLE 1. Spectral gaps of graphs generated by the GEO-P(Age) and the GEO-P(InvAge) model.

$N = 7115$, $\alpha = 0.7$, $\beta = 0.15$, $p = 1$			
GEO-P(Age)		**GEO-P(InvAge)**	
dimension	**gap**	**dimension**	**gap**
1	35.201192	1	0.913422
2	32.509603	2	129.049955
3	2.678022	3	32.13571
4	52.84162	4	48.921797
5	95.301035	5	96.451019

TABLE 2. Diameters of graphs generated by the GEO-P(Age) and the GEO-P(InvAge) model.

$N = 7115$, $\alpha = 0.7$, $\beta = 0.15$, $p = 1$			
GEO-P(Age)		**GEO-P(InvAge)**	
dimension	**diameter**	**dimension**	**diameter**
1	33	1	61
2	10	2	10
3	5	3	5
4	4	4	4
5	3	5	2

when the dimension increases; and the spectral gaps all decrease until the third dimension, then begin to increase. We may conclude that there is no significant difference between these models. In particular, all fit real OSNs well, and ranking by age does not provide a better fit to OSNs than the GEO-P model.

2. Ranking by Degree

The second group of OSN models we investigated are the GEO-P(Deg) model and the GEO-P(GnpDeg) model. These two models

have a new ranking system, where ranking is determined by the degrees of vertices. We assign larger degree vertex higher rank as it is plausible that more popular users in OSN may receive more invitations to become friends from other users. Hence, a higher degree vertex has a stronger ability of attracting new edges. If two vertices coincidentally have the same degree, then we take age as the second measurement to break ties.

The GEO-P(Deg) and the GEO-P(GnpDeg) models have four parameters: the attachment strength $\alpha \in (0, 1)$, the density parameter $\beta \in (0, 1-\alpha)$, the dimension $m \in \mathbb{N}$, and the link probability $p \in (0, 1]$. Let S be unit hypercube in $\mathbb{R}^m$, and $[N]$ be integers set from 1 to N. We describe the evolution process of the GEO-P(Deg) model:

1. At the initial time-step $t = 0$, we choose N vertices in S at random, randomly assign each vertex a unique rank chosen from the set $[N]$, and set all vertices ages as 0. An edge between a pair of vertices x and y is created with probability p if

$$d(x, y) \leq \max\{\mathrm{Rad}(x), \mathrm{Rad}(y)\}.$$

   We record degrees of all vertices at time-step $t = 0$.

2. At time-step $t > 0$, a randomly chosen vertex $v \in V(G_{t-1})$ dies, we remove all edges connected to v. For remaining vertices, if $vx \in$

$E(G_{t-1})$, then we update their degrees by

$$\deg_{G_t}(x) = \deg_{G_{t-1}}(x) - 1,$$

where $x \in V(G_{t-1}) \setminus \{v\}$. On the other hand, a new vertex u is born, we assign its age as t, and assign its degree and rank both 0. We now determine edges between every pair of vertices based on their ranks at time-step $t-1$. In the last step, we update and record vertices' ranks at time-step t by as follows.

$$K = \mid \{y \in V(G_t), \deg_{G_t}(y) > \deg_{G_t}(x)\} \mid, \tag{13}$$

$$L = \mid \{y \in V(G_t), \deg_{G_t}(y) = \deg_{G_t}(x) \text{ and } \mathrm{age}(y) < \mathrm{age}(x)\} \mid, \tag{14}$$

$$r(x,t) = K + L + 1. \tag{15}$$

In (13), (14), and (15), K indicates the number of vertices which degrees are bigger than vertex x, L indicates the number of vertices which degree are equal to vertex x, but older than vertex x. Hence, $K + L$ is the number of vertices whose ranks are higher than vertex x. Note that only one vertex is born at every time-step t, so we will not have duplicated ranks.

The GEO-P(GnpDeg) model is similar to the GEO-P(Deg) model, the only difference is the GEO-P(GnpDeg) model starts from a random

graph other than a GEO-P graph at time-step $t = 0$. The GEO-P(GnpDeg) models generates graphs by following process:

1. At initial time-step $t = 0$, we choose N vertices in S at random and generate a random graph $G(N, p)$. Thus, we determine edges between every pair of vertices by a fixed probability. Then we apply Formula 13, 14 and 15 to calculate vertices' ranks. Ages of all vertices are set to 0.
2. At time-step $t > 0$, a randomly chosen vertex $v \in V(G_{t-1})$ dies, we remove all edges connected to v. For remaining vertices, if $vx \in E(G_{t-1})$, then we update their degrees by

$$\deg_{G_t}(x) = \deg_{G_{t-1}}(x) - 1,$$

where $x \in V(G_{t-1}) \setminus \{v\}$. On the other hand, a new vertex u is born, we assign its age as t, and assign its degree and rank both 0. We now determine edges between every pair of vertices based on their ranks at time-step $t - 1$. The last step, we update and record vertices' ranks at time-step t by formulas (13), (14) and (15).

Note that in graphs generated by the GEO-P(GnpDeg) model, vertices ranks may not be unique, because vertices' ranks at time-step

$t = 0$ are determined by degrees, we may have some vertices with the same ranks and ages, which may cause a tie in following time-step t.

Now we analyse degree distribution, diameter, spectral gap of graphs generated by the GEO-P(Deg) and the GEO-P(GnpDeg) models. We applied the GEO-P(Deg) and the GEO-P(GnpDeg) models to generated graphs in dimensions 1 to 5, and 1 to 4, respectively. Figures 4.3 and 4.4 display the degree distribution log-log plot. Tables 4 and 3 list the diameters and gaps. We can see that the GEO-P(Deg) and the GEO-P(GnpDeg) models have different properties. Graphs generated by the GEO-P(Deg) model have similar properties as other GEO-P-type models. But graphs generated by the GEO-P(GnpDeg) have some different properties. Their degree distribution does not follow a power law distribution, have a larger spectral gap comparing to other GEO-P-type models, and their diameters do not shrink when the dimension increases. Their behaviors are closer to random graphs and this model is not a good one for OSNs.

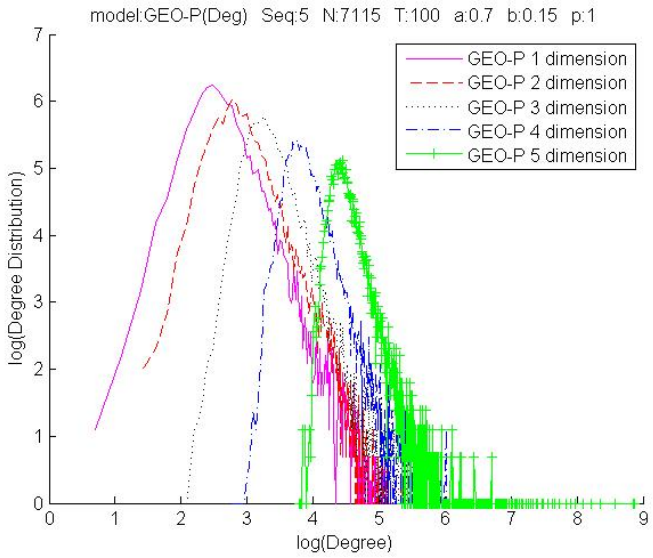


FIGURE 4.3. Degree distribution of graph generated by the GEO-P(Deg) model.

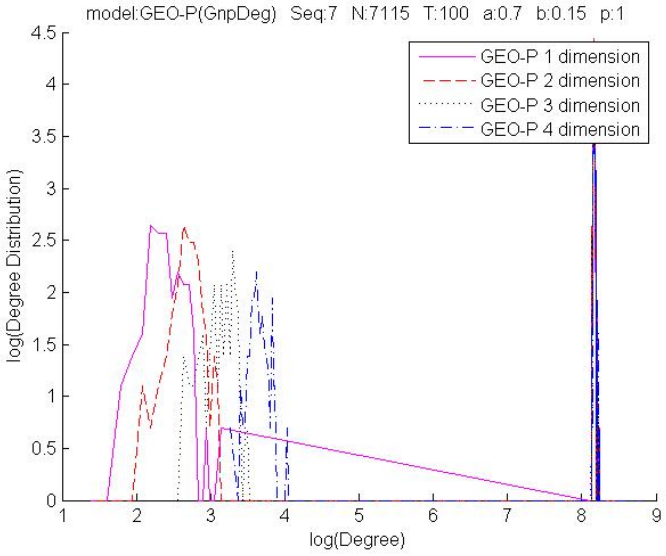


FIGURE 4.4. Degree distribution of graph generated by the GEO-P(GnpDeg) model.

TABLE 3. Spectral gaps of graphs generated by the GEO-P(Deg) and the GEO-P(GnpDeg) model.

$N = 7115$, $\alpha = 0.7$, $\beta = 0.15$, $p = 1$			
GEO-P(Deg)		**GEO-P(GnpDeg)**	
dimension	**gap**	**dimension**	**gap**
1	27.508783	1	3440.284205
2	17.59291	2	3443.40596
3	17.618218	3	3450.151086
4	94.816041	4	3466.83475
5	112.3931		

TABLE 4. Diameters of graphs generated by the GEO-P(Deg) and the GEO-P(GnpDeg) model.

$N = 7115$, $\alpha = 0.7$, $\beta = 0.15$, $p = 1$			
GEO-P(Deg)		**GEO-P(GnpDeg)**	
dimension	**diameter**	**dimension**	**diameter**
1	30	1	3
2	9	2	3
3	5	3	3
4	4	4	3
5	2		

3. Tension Parameter

Figure 4.5 shows the influence regions of vertices 5, 7 and 16. We see that vertices 7 and 16 both fall in the influence region of vertex 5. Based on the GEO-P model, the pairs of vertices 5, 7 and 5, 16 may both obtain edges (let us assume the link probability parameter $p = 1$). But on the other hand, we see that vertex 7 has a larger influence region then vertex 16, and correspondingly vertex 7 should have stronger ability of attracting edges. Hence, the overall ability of

obtaining edge of vertices 5 and 7 should be stronger than vertices 5 and 16.

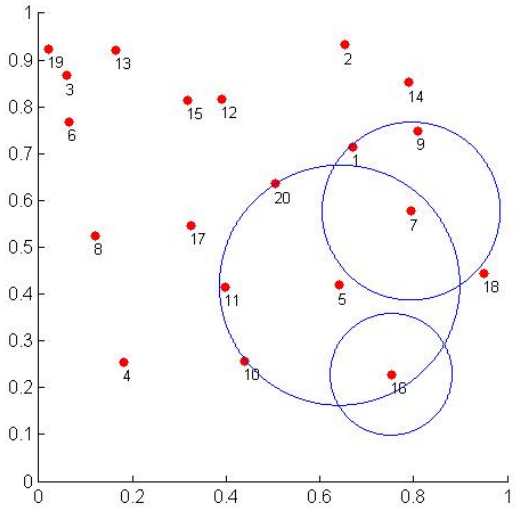


FIGURE 4.5. Influence region of vertices 5, 7 and 16.

We introduce *tension parameter* $h \in (-\infty, 0)$. Recall that the influence region is determined by the rank of vertex; that is, a higher rank vertex has stronger ability of obtaining edges than lower rank vertex. We define the *tension probability*, written

$$TP(x,t) = r(x,t)^h,$$

where x is a given vertex, and $r(x,t)$ is the rank of vertex x at time-step t. Note that $r(x,t) \geq 1$, so $TP(x,t) \in (0,1)$. We use

$$\frac{TP(x,t) + TP(y,t)}{2}$$

to donate the overall tension probability of vertices x and y.

Based on the tension parameter, we introduce the GEO-P(Ten) model. This model has four parameters: the attachment strength $\alpha \in (0,1)$, the density parameter $\beta \in (0, 1-\alpha)$, the dimension $m \in \mathbb{N}$, and the tension parameter $h \in (-\infty, 0)$. The GEO-P(Ten) model is the same as the GEO-P model, except for how to determine edges between every pair of vertices. Let S be unit hypercube in $\mathbb{R}^m$, and $[N]$ be integers set from 1 to N. The GEO-P(Ten) model generates graphs by following process:

1. At the initial time-step $t = 0$, we randomly choose N vertices in S, randomly assign vertex unique rank chosen from set $[N]$. An edge between pair of vertices x and y is created, if

$$d(x,y) \leq \max(\mathrm{Rad}(x), \mathrm{Rad}(y)) \left(\frac{TP(x,t) + TP(y,t)}{2} \right). \tag{16}$$

2. At time-step $t > 0$, a randomly chosen vertex $v \in V(G_{t-1})$ dies, a new vertex u is born and is assign a rank randomly chosen from set

$[N]$. We update ranks of vertices $x \in V(G_{t-1}) \cap V(G_t)$ by (2), then evaluate edges by (16)

Now we analyse degree distribution, diameter, spectral gap of graphs generated by the GEO-P(Ten) model. We applied the GEO-P(Ten) model to generated graphs in dimensions 1 to 5 inclusive, and we choose different values for tension parameter: $h = -0.1$, $h = -0.3$ and $h = -0.7$. Figures 4.6, 4.7 and 4.8 display the degree distribution log-log plots. Tables 6 and 5 list the diameters and gaps.

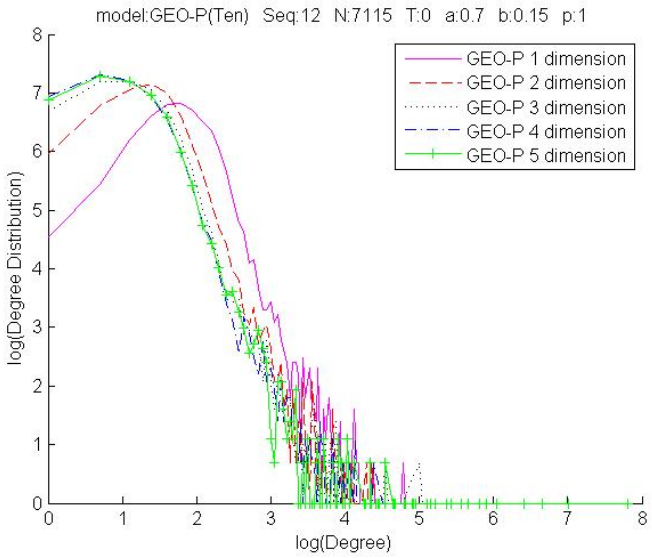


FIGURE 4.6. Degree distribution of graph generated by the GEO-P(Ten) model, with tension parameter $h = -0.1$.

We can see that graphs generated by the GEO-P(Ten) model have small spectral gaps and diameters, which are almost of the same order of the GEO-P model. From Figures 4.6, 4.7, and 4.8, we can see

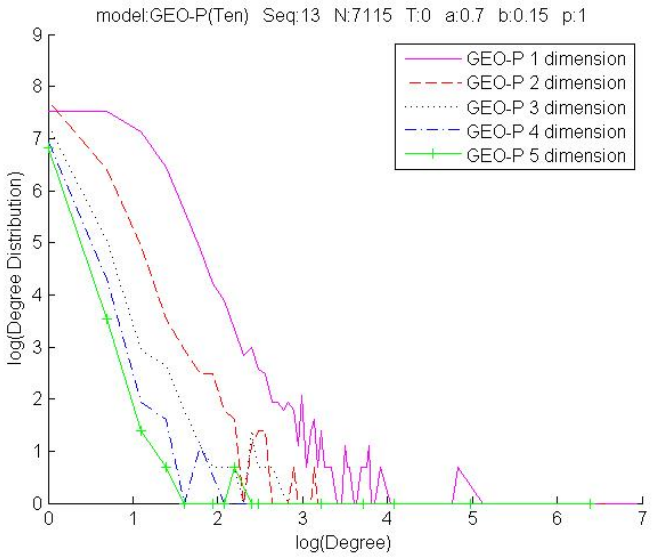


FIGURE 4.7. Degree distribution of graph generated by the GEO-P(Ten) model, with tension parameter $h = -0.3$.

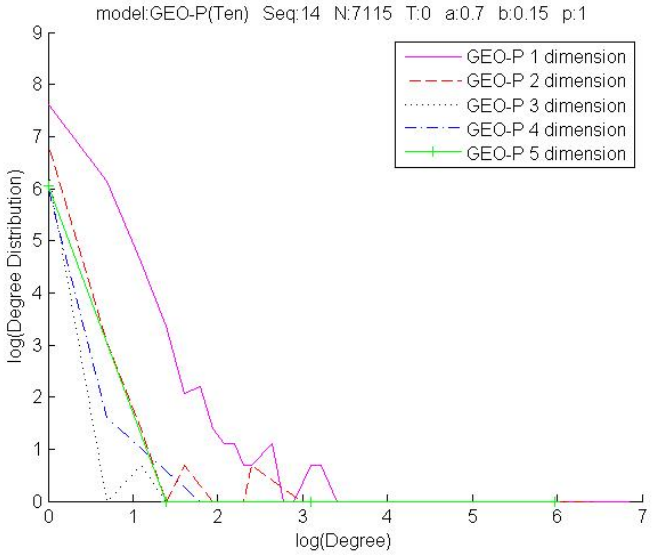


FIGURE 4.8. Degree distribution of graph generated by the GEO-P(Ten) model, with tension parameter $h = -0.7$.

TABLE 5. Spectral gaps of graphs generated by the GEO-P(Ten) model, in dimensions 1 to 5 inclusive.

GEO-P(Ten) $N = 7115$, $\alpha = 0.7$, $\beta = 0.15$, $p = 1$					
$h = -0.1$		$h = -0.3$		$h = -0.7$	
dim	**gap**	**dim**	**gap**	**dim**	**gap**
1	14.170589	1	9.048678	1	13.048242
2	0	2	9.286437	2	12.406163
3	11.410871	3	10.354512	3	13.150765
4	15.54475	4	10.186001	4	14.226928
5	20.845548	5	12.422439	5	15.133812

TABLE 6. Diameters of graphs generated by the GEO-P(Ten) model.

GEO-P(Ten) $N = 7115$, $\alpha = 0.7$, $\beta = 0.15$, $p = 1$						
	$h = -0.1$		$h = -0.3$		$h = -0.7$	
dimension	**diam**	$\mid V(C) \mid$	**diam**	$\mid V(C) \mid$	**diam**	$\mid V(C) \mid$
1	20	7098	6	6300	4	2673
2	20	6953	7	3149	4	990
3	20	6847	7	1648	3	571
4	15	6744	6	1152	3	415
5	12	6747	5	977	2	428

that graphs generated by the GEO-P(Ten) model follow a power law degree distribution in a larger range than the GEO-P model. Our initial results indicate that the GEO-P(Ten) model accurately fits OSN data as well as the GEO-P model.

CHAPTER 5

Conclusion and Open Problems

We introduced OSNs and verified graph theoretic properties of OSNs. Our computational results provide support for observed properties of OSNs: namely, power law degree distributions, the small world property, and bad spectral expansion.

The simulation results in sections 3.1 and 3.3 support the Logarithmic Dimension Hypothesis, which conjectures that the dimension of OSNs is

$$m = \lceil \log N \rceil,$$

where N is the order of the network. We compared the spectral gap of OSN samples and graphs generated by the GEO-P model in different dimensions, and our results suggest that the best fitting dimension is about $\log N$. We also compared the diameter of OSN samples and graphs generated by the GEO-P model, and found this was best fit to real OSN data when the dimension was about $\log N$.

We also introduced variants of GEO-P-type models which naturally simulate OSNs. We used the models to simulate OSNs, and verify how they fit OSNs by calculating the three main properties. Table 1

summarizes results from our simulations. According to that table, most of the GEO-P-type models fit OSNs well, and the degree distribution of graphs generated by the GEO-P(Ten) model follow a power law degree distribution for a bigger range of vertices. Only the GEO-P(GnpDeg) model does not fit OSNs well because of it does not seem to generate power law graphs.

TABLE 1. The GEO-P-type models simulation and calculation results

model	**power law**	**small wold**	**spectral gap**
GEO-P	**yes**	**yes**	**small**
GEO-P(Age)	**yes**	**yes**	**small**
GEO-P(InvAge)	**yes**	**yes**	**small**
GEO-P(Deg)	**yes**	**yes**	**small**
GEO-P(GnpDeg)	**no**	**yes**	**large**
GEO-P(Ten)	**yes**	**yes**	**small**

Several open problems arose while this research was conducted. We hope to be able to address these problems in future work. We collect and state these problems below.

1. OSNs usually have large orders, with many millions of vertices and edges. Owing to computational bottlenecks, we chose to work with smaller samples of OSNs, with orders in the thousands of vertices and edges. Higher order samples or even a whole OSN sample will likely give a more accurate result.

2. In Chapters 3 and 4, we only evaluate diameters of graphs to verify the small word property. The clustering coefficient is also a measurement of the small world property. Calculating clustering coefficients is another step to verify the small world property in both the models and the sampled OSN data. Tuning the models to give the correct clustering coefficient may give more evidence for the LDH.
3. For the GEO-P model, we have a few parameters, such as: the attachment strength, the density parameter, the dimension, and the link probability. How do we best choose these parameters?
4. In our OSN models, we use consecutive integers to assign vertices ranks, which leads to similar influence regions at different time-step t. Assigning vertices with positive real number as ranks may be more natural. One idea is that vertices ranks could be determined based on their popularity. We calculate the *degree ratio* for every vertex, which is defined as

$$RA(x,t) = \frac{\deg_{G_t}(x)}{\sum_{v \in V(G_t)} \deg_{G_t}(v)}.$$

Hence, the degree ratio indicates how popular the corresponding user is; that is, a bigger degree ratio refers to a more popular user. Note that since $RA(x,t) \in [0,1)$, we can then define rank function

based on the degree ratio. For example, we may define

$$R(x,T) = RA(x,t)^k,$$

where k is a negative real number.

5. For the GEO-P Model, rigorous proofs were given that the model a.a.s. generates graphs with a power law degree distribution, low diameter, high clustering coefficient, and bad spectral expansion. See Chapter 2, Theorems 2.1, 2.2, 2.3, 2.4, and 2.5 for the rigorous results for the GEO-P model. Can we rigorously prove with probability tending to 1 as n and t tend to infinity, that the models for OSNs presented in this thesis satisfy the desired properties?
6. From our simulation results, the GEO-P(GnpDeg) model does not fit the real OSN data well. However, we may need to run the time t much longer for the model to generate graphs with the desired properties.

CHAPTER 6

Appendix

The following original code of mysql database and matlab was used to simulate OSNs and the various stochastic models presented in this thesis. Further, we provide the code used to mine the real OSN data for various properties.

```
1
2  -- ----------------------------
3  -- Table structure for geopdegdist
4  -- ----------------------------
5  DROP TABLE IF EXISTS `geopdegdist`;
6  CREATE TABLE `geopdegdist` (
7    `seq` int(11) default NULL,
8    `degree` int(11) default NULL,
9    `dist` int(11) default NULL,
10   `flag` varchar(10) default NULL,
11   `N` int(11) default NULL,
12   `dim` int(11) default NULL,
13   `a` decimal(10,2) default NULL,
14   `b` decimal(10,2) default NULL,
15   `T` int(11) default NULL
16 ) ENGINE=MyISAM DEFAULT CHARSET=latin1;
17
18 -- ----------------------------
19 -- Table structure for geopsample
20 -- ----------------------------
21 DROP TABLE IF EXISTS `geopsample`;
22 CREATE TABLE `geopsample` (
23   `seq` int(11) NOT NULL default '0',
24   `userid` int(11) NOT NULL default '0',
25   `userlabel` int(11) default NULL,
26   `outdegree` int(11) default NULL,
27   `indegree` int(11) default NULL,
28   `u` int(11) default '0',
29   `N` int(11) NOT NULL default '0',
30   `dim` int(11) NOT NULL default '0',
31   `a` decimal(10,2) NOT NULL default '0.00',
32   `b` decimal(10,2) NOT NULL default '0.00',
33   `T` int(11) NOT NULL default '0',
```

```
34    PRIMARY KEY  (`seq`,`userid`,`N`,`a`,`b`,`dim`,`T`),
35    KEY `ind_userid` USING BTREE (`userid`,`seq`)
36  ) ENGINE=MyISAM DEFAULT CHARSET=latin1;
37
38  -- ----------------------------
39  -- Table structure for geopsample_links
40  -- ----------------------------
41  DROP TABLE IF EXISTS `geopsample_links`;
42  CREATE TABLE `geopsample_links` (
43    `userid` int(11) NOT NULL default '0',
44    `friendid` int(11) NOT NULL default '0',
45    `userlabel` int(11) default NULL,
46    `friendlabel` int(11) default NULL,
47    `seq` int(11) NOT NULL default '0',
48    `dim` int(11) NOT NULL default '0',
49    `N` int(11) NOT NULL default '0',
50    `a` decimal(10,2) NOT NULL default '0.00',
51    `b` decimal(10,2) NOT NULL default '0.00',
52    `T` int(11) NOT NULL default '0',
53    PRIMARY KEY  (`seq`,`userid`,`friendid`,`N`,`a`,`b`,`dim`,`T`),
54    KEY `ind_userid` USING BTREE (`userid`,`seq`,`dim`,`N`,`a`,`b`),
55    KEY `ind_friendid` USING BTREE (`friendid`,`seq`,`dim`,`N`,`a`,`b`)
56  ) ENGINE=MyISAM DEFAULT CHARSET=latin1;
57
58  -- ----------------------------
59  -- Table structure for geopsampleN
60  -- ----------------------------
61  DROP TABLE IF EXISTS `geopsampleN`;
62  CREATE TABLE `geopsampleN` (
63    `seq` int(11) default '0',
64    `userid` int(11) default NULL,
65    `Nlevel` int(11) NOT NULL default '0',
66    `friendid` int(11) default NULL,
67    `N` int(11) default NULL,
68    `dim` int(11) default NULL,
69    `a` decimal(10,2) default NULL,
70    `b` decimal(10,2) default NULL,
71    `T` int(11) default NULL,
72    KEY `ind_userid` (`seq`,`userid`,`Nlevel`)
73  ) ENGINE=MyISAM DEFAULT CHARSET=latin1;
74
75  -- ----------------------------
76  -- Table structure for wiki_degdist
77  -- ----------------------------
78  DROP TABLE IF EXISTS `wiki_degdist`;
79  CREATE TABLE `wiki_degdist` (
80    `seq` int(11) default NULL,
81    `degree` int(11) default NULL,
82    `dist` int(11) default NULL,
83    `flag` varchar(10) default NULL
84  ) ENGINE=MyISAM DEFAULT CHARSET=latin1;
85
86  -- ----------------------------
```

```
87  -- Table structure for wikilinks
88  -- ----------------------------
89  DROP TABLE IF EXISTS `wikilinks`;
90  CREATE TABLE `wikilinks` (
91    `userid` int(11) default NULL,
92    `friendid` int(11) default NULL
93  ) ENGINE=MyISAM DEFAULT CHARSET=latin1;
94
95  -- ----------------------------
96  -- Table structure for wikisample
97  -- ----------------------------
98  DROP TABLE IF EXISTS `wikisample`;
99  CREATE TABLE `wikisample` (
100   `seq` int(11) NOT NULL default '0',
101   `userid` int(11) NOT NULL default '0',
102   `userlabel` int(11) default NULL,
103   `outdegree` int(11) default NULL,
104   `indegree` int(11) default NULL,
105   `u` int(11) default '0',
106   PRIMARY KEY  (`seq`,`userid`),
107   KEY `ind_userid` USING BTREE (`userid`,`seq`)
108 ) ENGINE=MyISAM DEFAULT CHARSET=latin1;
109
110 -- ----------------------------
111 -- Table structure for wikisample_links
112 -- ----------------------------
113 DROP TABLE IF EXISTS `wikisample_links`;
114 CREATE TABLE `wikisample_links` (
115   `userid` int(11) NOT NULL default '0',
116   `friendid` int(11) NOT NULL default '0',
117   `userlabel` int(11) default NULL,
118   `friendlabel` int(11) default NULL,
119   `seq` int(11) NOT NULL default '0',
120   PRIMARY KEY  (`userid`,`friendid`,`seq`),
121   KEY `ind_userid` USING BTREE (`userid`,`seq`),
122   KEY `ind_friendid` USING BTREE (`friendid`,`seq`)
123 ) ENGINE=MyISAM DEFAULT CHARSET=latin1;
124
125 -- ----------------------------
126 -- Table structure for wikisampleN
127 -- ----------------------------
128 DROP TABLE IF EXISTS `wikisampleN`;
129 CREATE TABLE `wikisampleN` (
130   `seq` int(11) default '0',
131   `userid` int(11) default NULL,
132   `N` int(11) NOT NULL default '0',
133   `friendid` int(11) default NULL,
134   KEY `ind_userid` (`seq`,`userid`,`N`)
135 ) ENGINE=MyISAM DEFAULT CHARSET=latin1;
136
137 -- ----------------------------
138 -- Table structure for youtube_degree
139 -- ----------------------------
```

```
DROP TABLE IF EXISTS `youtube_degree`;
CREATE TABLE `youtube_degree` (
  `userid` int(20) NOT NULL,
  `indegree` int(20) NOT NULL,
  `outdegree` int(20) NOT NULL,
  PRIMARY KEY  (`userid`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

-- ----------------------------
-- Table structure for youtubedegree_dist
-- ----------------------------
DROP TABLE IF EXISTS `youtubedegree_dist`;
CREATE TABLE `youtubedegree_dist` (
  `degree` int(11) default NULL,
  `dist` int(11) default NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

-- ----------------------------
-- Table structure for youtubelinks
-- ----------------------------
DROP TABLE IF EXISTS `youtubelinks`;
CREATE TABLE `youtubelinks` (
  `userid` int(11) default NULL,
  `friendid` int(11) default NULL,
  KEY `Ind_youtubelinks` (`userid`),
  KEY `ind_youtubelinks2` (`friendid`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

-- ----------------------------
-- Table structure for youtubelinks_degree
-- ----------------------------
DROP TABLE IF EXISTS `youtubelinks_degree`;
CREATE TABLE `youtubelinks_degree` (
  `userid` int(11) default NULL,
  `degree` int(11) default NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

-- ----------------------------
-- Table structure for youtubesample
-- ----------------------------
DROP TABLE IF EXISTS `youtubesample`;
CREATE TABLE `youtubesample` (
  `seq` int(5) NOT NULL,
  `source` varchar(50) default NULL,
  `userid` int(11) NOT NULL,
  `userlabel` int(11) default NULL,
  `outdegree` int(11) default NULL,
  `indegree` int(11) default NULL,
  `u` int(11) default '0',
  PRIMARY KEY  (`seq`,`userid`),
  KEY `ind_userid` (`seq`,`userid`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

```

```
193 -- ----------------------------
194 -- Table structure for youtubesample_links
195 -- ----------------------------
196 DROP TABLE IF EXISTS `youtubesample_links`;
197 CREATE TABLE `youtubesample_links` (
198   `userid` int(11) NOT NULL,
199   `friendid` int(11) default NULL,
200   `userlabel` int(11) default NULL,
201   `friendlabel` int(11) default NULL,
202   `seq` int(11) NOT NULL,
203   KEY `ind_userid` (`seq`,`userid`),
204   KEY `ind_friendid` (`seq`,`friendid`)
205 ) ENGINE=MyISAM DEFAULT CHARSET=latin1;
206
207 -- ----------------------------
208 -- Table structure for youtubesampleN
209 -- ----------------------------
210 DROP TABLE IF EXISTS `youtubesampleN`;
211 CREATE TABLE `youtubesampleN` (
212   `seq` int(11) default NULL,
213   `userid` int(11) default NULL,
214   `N` int(11) default NULL,
215   `friendid` int(11) default NULL
216 ) ENGINE=MyISAM DEFAULT CHARSET=latin1;
217
218
219
220 -- ----------------------------
221 -- procedure structure for proc_geopsample
222 -- generate geop sample user information from 1 to Nnum (update table
       geopsample), calcuate indegree and outdegree (update table
       geopsample), calculate degree distribution (update table
       geopdegdist)
223 -- ----------------------------
224 CREATE DEFINER = `amandatian`@`%` PROCEDURE `proc_geopsample`(in Seq
       int, in Nnum int, in Dim int, in A decimal(5,2), in B decimal(5,2)
       , in Ttime int)
225 BEGIN
226 /* generate geop sample user information from 1 to Nnum (update table
       geopsample), calcuate indegree and outdegree (update table
       geopsample), calculate degree distribution (update table
       geopdegdist)*/
227
228 DECLARE N1 int;
229 DECLARE N2 int;
230 DECLARE Exsist int default 0;
231 DECLARE rowstatus int default 0;
232 DECLARE i int default 1;
233
234
235 DECLARE User int;
236 DECLARE degree int;
237 DECLARE num int default 0;
```

```
238 DECLARE U int;
239
240 DECLARE csr_user2 CURSOR FOR select geopsample.userid from geopsample
        where geopsample.seq = Seq and geopsample.N=Nnum and geopsample.
        dim = Dim and geopsample.a = A and geopsample.b = B  and
        geopsample.T = Ttime;
241 DECLARE CONTINUE HANDLER FOR NOT FOUND SET rowstatus=1;
242
243 delete from geopsample where geopsample.seq = Seq and geopsample.N=
        Nnum and geopsample.dim = Dim and geopsample.a = A and geopsample.
        b = B  and geopsample.T = Ttime;
244
245 /* generate geop sample user information from 1 to Nnum */
246 while i <= Nnum do
247         insert into `geopsample` (userid,userlabel,seq,N,dim,a,b,T)
                values ( i, i,Seq,Nnum,Dim,A,B,Ttime);
248         set i = i + 1;
249
250 end while;
251 commit;
252
253
254 /* calculate degree*/
255 set rowstatus=0;
256
257 OPEN csr_user2;
258 FETCH csr_user2 INTO User;
259 WHILE  rowstatus=0  DO
260         select count(*) into degree from geopsample_links where
                geopsample_links.userid = User and geopsample_links.seq =
                Seq and geopsample_links.N=Nnum and geopsample_links.dim =
                 Dim and geopsample_links.a = A and geopsample_links.b = B
                  and geopsample_links.T = Ttime ;
261
262         if degree = 0 then
263                 set U = 1;
264         else
265                 set U= 0;
266         end if;
267
268         set num = num +1;
269
270         update geopsample set geopsample.outdegree = degree ,
                geopsample.indegree = degree , geopsample.u = U where
                geopsample.userid = User and  geopsample.seq = Seq and
                geopsample.N=Nnum and geopsample.dim = Dim and geopsample.
                a = A and geopsample.b = B  and geopsample.T = Ttime;
271
272         FETCH csr_user2 INTO User;
273 END WHILE;
274 CLOSE csr_user2;
275 commit;
276
```

```
277 /* calculate degree distribution*/
278 delete from geopdegdist where geopdegdist.seq = Seq and geopdegdist.N=
        Nnum and geopdegdist.dim = Dim and geopdegdist.a = A and
        geopdegdist.b = B  and geopdegdist.T = Ttime;
279
280 insert into geopdegdist ( flag, seq, N, dim, a, b, T, degree, dist)
281 select    '', geopsample.seq, geopsample.N, geopsample.dim, geopsample
        .a, geopsample.b, geopsample.T, geopsample.indegree, count(*) from
         geopsample
282 where geopsample.seq = Seq and geopsample.N=Nnum and geopsample.dim =
        Dim and geopsample.a = A and geopsample.b = B and geopsample.T =
        Ttime
283 group by geopsample.indegree ;
284
285 END;
286
287
288 -- ----------------------------
289 -- procedure structure for proc_geopsamplecall
290 -- to generate geop sample information and geop degree distribution
        data for multiple dimensions
291 -- ----------------------------
292 CREATE DEFINER = `amandatian`@`%` PROCEDURE `proc_geopsamplecall`(in
        Seq int, in num int, in A decimal(5,2), in B decimal(5,2), in DimS
         int, in DimE int, in Ttime int)
293 BEGIN
294 /* to generate geop sample information and geop degree distribution
        data for multiple dimensions*/
295 Declare i int;
296 set i = DimS;
297 while i <= DimE do
298         call proc_geopsample(Seq, num, i , A, B, Ttime);
299         set i = i +1;
300 end while;
301 END;
302
303
304
305 -- ----------------------------
306 -- procedure structure for proc_geopsampleN
307 -- generate neighbour lists for some given vertices, used to calculate
         diameter, update table geopsampleN
308 -- ----------------------------
309 CREATE DEFINER = `amandatian`@`%` PROCEDURE `proc_geopsampleN`(in Seq
        int, in Nnum int, in Dim int, in A decimal(5,2), in B decimal(5,2)
        , in Ttime int)
310 BEGIN
311 /* generate neighbour lists for some given vertices, used to calculate
         diameter, update table geopsampleN */
312 DECLARE num int default 0;
313 DECLARE Neigh int;
314 DECLARE NPare int;
315 DECLARE insertCnt int;
```

```
316 DECLARE PareUserCnt int default 0;
317 DECLARE strLength int;
318 DECLARE Nparelist varchar(10000);
319 DECLARE Exsist1 int default 0;
320 DECLARE Exsist2 int default 0;
321 DECLARE PareUser int DEFAULT 1;
322 DECLARE rowstatus int default 0;
323 DECLARE GoStatus int default 0;
324 DECLARE tbl varchar(40);
325 DECLARE sqldel varchar(1000);
326 DECLARE sqlinsert1 varchar(1000);
327 DECLARE sqlsel1 varchar(1000);
328 DECLARE sqlsel2 varchar(1000);
329
330 /*  declare cursor for every node in geop sample*/
331 DECLARE csr_user0 CURSOR FOR select geopsample.userid from geopsample
        where geopsample.userid<=2  and geopsample.seq = Seq  and
        geopsample.N= Nnum and geopsample.dim= Dim and geopsample.a= A and
         geopsample.b = B and geopsample.T = Ttime order by geopsample.
        userid ;
332 /* select all distinct neibourgh of level num of PareUser, using
        neibourgh list*/
333 DECLARE csr_user1 CURSOR FOR select distinct(geopsample_links.friendid
        ) from geopsample_links
334 where geopsample_links.seq = Seq and geopsample_links.N= Nnum and
        geopsample_links.dim= Dim and geopsample_links.a= A and
        geopsample_links.b = B and geopsample_links.T = Ttime
335 and FIND_IN_SET(geopsample_links.userid, Nparelist);
336 /* generage neibourgh list of level num of PareUser*/
337 DECLARE csr_user2 CURSOR FOR select distinct(geopsampleN.friendid)
        from geopsampleN
338 where geopsampleN.seq = Seq and geopsampleN.N= Nnum and geopsampleN.
        dim= Dim and geopsampleN.a= A and geopsampleN.b = B and
        geopsampleN.T = Ttime
339 and geopsampleN.userid=PareUser and geopsampleN.Nlevel=num;
340 DECLARE CONTINUE HANDLER FOR NOT FOUND SET rowstatus=1;
341
342 set tbl = 'geopsampleN';
343 select Seq;
344 select Nnum;
345 select A;
346 select B;
347 select Dim;
348
349 set @sqldel = concat('delete from ', tbl, ' where ', tbl, '.seq = ',
        Seq,  ' and ', tbl, '.N=', Nnum, ' and ', tbl, '.dim=', Dim, ' and
         ', tbl, '.a=' , A, ' and ', tbl, '.b=', B , ' and ', tbl, '.T=',
        Ttime   );
350 PREPARE stmt FROM @sqldel ;
351 EXECUTE stmt;
352 DEALLOCATE PREPARE stmt;
353
354 OPEN csr_user0;
```

```
355 FETCH csr_user0 INTO PareUser;
356 WHILE rowstatus=0 DO
357         /*initial all counter or parameters*/
358         set PareUserCnt = PareUserCnt +1;
359         set num = 0;
360         set Nparelist = PareUser;
361         set GoStatus = 0;
362         /* insert the node itself as level 0 neighbour*/
363         insert into geopsampleN (seq, N, dim, a, b, T, userid, Nlevel,
                friendid) values (Seq, Nnum, Dim, A, B, Ttime, PareUser,
               num, Pareuser);
364
365         /* do the loop before all new level neighbours have been found
               */
366         While GoStatus = 0 Do
367                 set num = num +1;
368                 set insertCnt = 0;
369                 OPEN csr_user1;
370                 FETCH csr_user1 INTO Neigh;
371                 WHILE rowstatus=0 DO
372
373                         /* verify if this neighbour has been insert
                               into the table*/
374                         set @sqlsel1 = concat('select count(*) into
                               @Exsist1 from ', tbl, ' where ',tbl, '.
                               friendid =', Neigh, ' and ', tbl, '.userid
                                = ', PareUser,  ' and ', tbl, '.seq = ',
                               Seq ,  ' and ', tbl, '.N=', Nnum, ' and ',
                                tbl, '.dim=', Dim, ' and ', tbl, '.a=' ,
                               A, ' and ', tbl, '.b=', B , ' and ', tbl,
                               '.T=', Ttime  );
375                         PREPARE stmt FROM @sqlsel1 ;
376                         EXECUTE stmt;
377                         DEALLOCATE PREPARE stmt;
378
379                         if @Exsist1=0 then
380                                 set @sqlinsert1 = concat('insert into
                                        ', tbl, '  (seq,  N, dim, a, b, T,
                                         userid, Nlevel, friendid) values
                                        (', Seq, ', ', Nnum, ', ', Dim, ',
                                         ', A, ', ', B, ', ' , Ttime, ', '
                                         , PareUser, ', ' , num, ', ',
                                        Neigh,  ')');
381                                 PREPARE stmt FROM @sqlinsert1 ;
382                                 EXECUTE stmt;
383                                 DEALLOCATE PREPARE stmt;
384                                 set insertCnt = insertCnt +1;
385                         end if;
386
387                         set @Exsist1 = 0;
388                         FETCH csr_user1 INTO Neigh;
389                 END WHILE;
```

```
390                 /*stop the loop when no new level  neighbour can be
                        found*/
391                 if insertCnt = 0 then
392                         set GoStatus = 1;
393                 end if;
394                 CLOSE csr_user1;
395                 set rowstatus =0 ;
396                 /* generate neighbour list for level num, will be used
                         at next round of loop*/
397                 set Nparelist = '';
398                 OPEN csr_user2;
399                 FETCH csr_user2 INTO NPare;
400                 WHILE rowstatus=0 DO
401                         set Nparelist = concat(Nparelist,  Npare, ',')
                            ;
402                         FETCH csr_user2 INTO NPare;
403                 END WHILE;
404                 CLOSE csr_user2;
405
406                 set rowstatus =0 ;
407                 set strLength = char_length(Nparelist);
408                 set Nparelist = substring(NpareList, 1, strLength-1);
409
410         END WHILE;
411         commit;
412         FETCH csr_user0 INTO PareUser;
413 END WHILE;
414 CLOSE csr_user0;
415
416
417 select Dim;
418
419 END;
420
421 -- ----------------------------
422 -- procedure structure for proc_wiki_degdist
423 -- calculate wiki user indegree and outdegree, alldegree (update table
        wiki_degdist)
424 -- ----------------------------
425 CREATE DEFINER = `amandatian`@`%` PROCEDURE `proc_wiki_degdist`()
426 BEGIN
427 /*  calculate wiki user indegree and outdegree, alldegree (update
       table wiki_degdist)*/
428 delete from wiki_degdist;
429 insert into wiki_degdist (seq, flag, degree, dist)
430 select 0, 'in', wikisample.indegree, count(*) from wikisample group by
        wikisample.indegree;
431
432 insert into wiki_degdist (seq, flag, degree, dist)
433 select 0, 'out', wikisample.outdegree, count(*) from wikisample group
       by wikisample.outdegree;
434
435 insert into wiki_degdist (seq, flag, degree, dist)
```

```
436 select 0, 'all', wikisample.outdegree + wikisample.indegree , count(*)
        from wikisample group by wikisample.outdegree + wikisample.
       indegree;
437
438 END;
439
440
441 -- ----------------------------
442 -- procedure structure for proc_wikisample
443 -- create wiki user information, check userid and friendid, then take
       distinct  (update table wikisample)
444 -- ----------------------------
445
446 CREATE DEFINER = `amandatian`@`%` PROCEDURE `proc_wikisample`()
447 BEGIN
448 /* create wiki user information, check userid and friendid, then take
       distinct  (update table wikisample)*/
449 DECLARE N1 int;
450 DECLARE N2 int;
451 DECLARE Exsist int default 0;
452
453 DECLARE rowstatus int default 0;
454
455 DECLARE csr_user1 CURSOR FOR select distinct(userid) from `
       wikisample_links` order by userid;
456 DECLARE csr_user2 CURSOR FOR select distinct(friendid) from `
       wikisample_links` order by friendid;
457 DECLARE CONTINUE HANDLER FOR NOT FOUND SET rowstatus=1;
458
459 delete from `wikisample`;
460 OPEN csr_user1;
461 FETCH csr_user1 INTO N1;
462 WHILE rowstatus=0 DO
463         insert into `wikisample` (userid) values (N1);
464         FETCH csr_user1 INTO N1;
465 END WHILE;
466 CLOSE csr_user1;
467 SET rowstatus=0;
468
469 OPEN csr_user2;
470 FETCH csr_user2 INTO N2;
471 WHILE rowstatus=0 DO
472         select count(*) into Exsist from `wikisample` where userid =
              N2;
473         if Exsist = 0 then
474                 insert into `wikisample` (userid) values (N2);
475                 set Exsist = 0;
476         end if;
477         FETCH csr_user2 INTO N2;
478 END WHILE;
479 CLOSE csr_user2;
480
481 END;
```

```
482
483 -- ----------------------------
484 -- procedure structure for proc_wikisamplelinks
485 -- update wiki user lables in table wikisample_links
486 -- ----------------------------
487 CREATE DEFINER = `amandatian`@`%` PROCEDURE `proc_wikisamplelinks`()
488 BEGIN
489 /*update wiki user lables in table wikisample_links*/
490 update `wikisample_links` ,wikisample
491 set wikisample_links.userlabel = `wikisample`.userlabel
492 where wikisample_links.userid = wikisample.userid ;
493
494 update `wikisample_links` ,wikisample
495 set wikisample_links.friendlabel = `wikisample`.userlabel
496 where wikisample_links.friendid = wikisample.userid;
497
498 END;
499
500
501 -- ----------------------------
502 -- procedure structure for proc_wikisampleN
503 -- generate neighbour list for some given vertices, used to calculate
        diameter (update table wikisampleN)
504 -- ----------------------------
505 CREATE DEFINER = `amandatian`@`%` PROCEDURE `proc_wikisampleN`()
506 BEGIN
507 /*generate neighbour list for some given vertices, used to calculate
        diameter (update table wikisampleN)*/
508
509 DECLARE num int default 0;
510 DECLARE N int;
511 DECLARE NPare int;
512 DECLARE insertCnt int;
513 DECLARE PareUserCnt int default 0;
514 DECLARE strLength int;
515 DECLARE Nparelist varchar(10000);
516 DECLARE Exsist1 int default 0;
517 DECLARE Exsist2 int default 0;
518 DECLARE PareUser int DEFAULT 1;
519 DECLARE rowstatus int default 0;
520 DECLARE GoStatus int default 0;
521 DECLARE tbl varchar(40);
522 DECLARE sqldel varchar(1000);
523 DECLARE sqlinsert1 varchar(1000);
524 DECLARE sqlsel1 varchar(1000);
525 DECLARE sqlsel2 varchar(1000);
526
527 DECLARE Seq int;
528 /*  declare cursor for every node in wiki sample*/
529 DECLARE csr_user0 CURSOR FOR select wikisample.userid from wikisample
        where wikisample.seq = Seq  order by wikisample.userid ;
530 /* select all distinct neibourgh of level num of PareUser, using
        neibourgh list*/
```

```
531 DECLARE csr_user1 CURSOR FOR select distinct(wikisample_links.friendid
        ) from wikisample_links where wikisample_links.seq = Seq and
        FIND_IN_SET(wikisample_links.userid, Nparelist);
532 /* generage neibourgh list of level num of PareUser*/
533 DECLARE csr_user2 CURSOR FOR select distinct(wikisampleN.friendid)
        from wikisampleN where wikisampleN.seq = Seq and wikisampleN.
        userid=PareUser and wikisampleN.N=num;
534 DECLARE CONTINUE HANDLER FOR NOT FOUND SET rowstatus=1;
535
536 set tbl = 'wikisampleN';
537 set Seq = 0;
538
539
540 set @sqldel = concat('delete from ', tbl, ' where ', tbl, '.seq = ',
        Seq);
541 PREPARE stmt FROM @sqldel ;
542 EXECUTE stmt;
543 DEALLOCATE PREPARE stmt;
544
545 OPEN csr_user0;
546 FETCH csr_user0 INTO PareUser;
547 WHILE rowstatus=0 DO
548         /*initial all counter or parameters*/
549         set PareUserCnt = PareUserCnt +1;
550         select PareUserCnt;
551         set num = 0;
552         set Nparelist = PareUser;
553         set GoStatus = 0;
554         /* insert the node itself as level 0 neighbout*/
555         insert into wikisampleN (seq, userid, N, friendid) values (Seq
              , PareUser, num, Pareuser);
556
557         /* do the loop before all new level neighbours have been found
              */
558         While GoStatus = 0 Do
559                 set num = num +1;
560                 set insertCnt = 0;
561                 OPEN csr_user1;
562                 FETCH csr_user1 INTO N;
563                 WHILE rowstatus=0 DO
564
565                         /* verify if this neighbour has been insert
                              into the table*/
566                         set @sqlsel1 = concat('select count(*) into
                              @Exsist1 from ', tbl, ' where ',tbl, '.
                              friendid =', N, ' and ', tbl, '.userid =
                              ', PareUser,  ' and ', tbl, '.seq = ', Seq
                              );
567                         PREPARE stmt FROM @sqlsel1 ;
568                         EXECUTE stmt;
569                         DEALLOCATE PREPARE stmt;
570
571                         if @Exsist1=0 then
```

```
572                                 set @sqlinsert1 = concat('insert into
                                        ', tbl, '  (seq,  userid, N,
                                        friendid) values (', Seq, ', ',
                                        PareUser, ', ' , num, ', ',N,  ')
                                        ');
573                                 PREPARE stmt FROM @sqlinsert1 ;
574                                 EXECUTE stmt;
575                                 DEALLOCATE PREPARE stmt;
576                                 set insertCnt = insertCnt +1;
577                         end if;
578
579                         set @Exsist1 = 0;
580                         FETCH csr_user1 INTO N;
581                 END WHILE;
582                 /*stop the loop when no new level  neighbour can be
                        found*/
583                 if insertCnt = 0 then
584                         set GoStatus = 1;
585                 end if;
586                 CLOSE csr_user1;
587                 set rowstatus =0 ;
588                 /* generate neighbour list for level num, will be used
                         at next round of loop*/
589                 set Nparelist = '';
590                 OPEN csr_user2;
591                 FETCH csr_user2 INTO NPare;
592                 WHILE rowstatus=0 DO
593                         set Nparelist = concat(Nparelist,  Npare, ',')
                                ;
594                         FETCH csr_user2 INTO NPare;
595                 END WHILE;
596                 CLOSE csr_user2;
597
598                 set rowstatus =0 ;
599                 set strLength = char_length(Nparelist);
600                 set Nparelist = substring(NpareList, 1, strLength-1);
601
602         END WHILE;
603         FETCH csr_user0 INTO PareUser;
604 END WHILE;
605 CLOSE csr_user0;
606 commit;
607
608 END;
609
610 -- ----------------------------
611 -- procedure structure for proc_wikiSampleOrder
612 -- create user lable informaiton for wiki sample, update table
        wikisample
613 -- ----------------------------
614 CREATE DEFINER = `amandatian`@`%` PROCEDURE `proc_wikiSampleOrder`()
615 BEGIN
```

```
/*create user lable informaiton for wiki sample, update table
    wikisample*/
DECLARE rowstatus int DEFAULT 0;
DECLARE User int;
DECLARE Num int;
DECLARE csr_user CURSOR FOR
select userid from wikisample order by userid  ;
DECLARE CONTINUE HANDLER FOR NOT FOUND SET rowstatus=1;
SET Num=0;
OPEN csr_user;
FETCH csr_user INTO User;
while rowstatus=0 DO
        SET Num = Num+1;
       /* if User = 1156640 THEN
        select Num;
        end if;*/
        UPDATE `wikisample` SET userlabel = Num WHERE userid = User;
         FETCH csr_user INTO User;

END WHILE;
CLOSE csr_user;
END;

-- ----------------------------
-- procedure structure for proc_wikisampleU
-- calculate wiki sample user indegree and outdegree ( update table
    wikisample) , if outdegree is 0, then update table wikisample, set
     u=1, is userd for to calculate NL gap
-- ----------------------------
CREATE DEFINER = `amandatian`@`%` PROCEDURE `proc_wikisampleU`()
BEGIN
/*calculate wiki sample user indegree and outdegree ( update table
    wikisample) , if outdegree is 0, then update table wikisample, set
     u=1, is userd for to calculate NL gap */
DECLARE rowstatus int default 0;
DECLARE User int;
DECLARE Outdegree int;
DECLARE Indegree int;
DECLARE Seq int;
DECLARE num int default 0;

DECLARE csr_user CURSOR FOR select wikisample.userid from wikisample
    where wikisample.seq =Seq ;
DECLARE CONTINUE HANDLER FOR NOT FOUND SET rowstatus=1;
set Seq = 0;

OPEN csr_user;
FETCH csr_user INTO User;
WHILE  rowstatus=0  DO
        select count(*) into Outdegree from wikisample_links where
            wikisample_links.userid = User and wikisample_links.seq =
            Seq;
```

```
660         select count(*) into Indegree from wikisample_links where
                wikisample_links.friendid = User and wikisample_links.seq
                =Seq;
661         set num = num +1;
662
663         update wikisample set wikisample.outdegree = Outdegree ,
                wikisample.indegree = Indegree where wikisample.userid =
                User and  wikisample.seq = Seq;
664         if Outdegree = 0 then
665                 update wikisample set wikisample.u=1 where wikisample.
                        userid = User and wikisample.seq = Seq;
666         end if;
667         FETCH csr_user INTO User;
668 END WHILE;
669 CLOSE csr_user;
670 commit;
671 /*select num;*/
672
673
674 END;
675
676
677 -- ----------------------------
678 -- procedure structure for proc_youtubesample
679 -- generate youtube samples with smaller order by choosing first two
        level neighbours of given vertex (specified by degree), update
        table youtubesample
680 -- Seq: sequence
681 -- K: degree to choose given vertex
682 -- Pare: how many given vertices as pareanet vertices.
683 -- ----------------------------
684
685 CREATE DEFINER = `amandatian`@`%` PROCEDURE `proc_youtubesample`(in
        Seq int, in K int, in Pare int)
686 BEGIN
687 /*generate youtube samples with smaller order by choosing first two
        level neighbours of given vertex (specified by degree), update
        table youtubesample
688 Seq: sequence
689 K: degree to choose given vertex
690 Pare: how many given vertices as pareanet vertices.*/
691
692 DECLARE num int default 1;
693 DECLARE N1 int;
694 DECLARE N2 int;
695 DECLARE Exsist0 int default 0;
696 DECLARE Exsist1 int default 0;
697 DECLARE Exsist2 int default 0;
698 DECLARE PareUser int DEFAULT 1;
699 DECLARE rowstatus int default 0;
700 DECLARE tbl varchar(40);
701 DECLARE sqldel varchar(1000);
702 DECLARE sqlinsert0 varchar(1000);
```

```
703 DECLARE sqlinsert1 varchar(1000);
704 DECLARE sqlinsert2 varchar(1000);
705 DECLARE sqlsel1 varchar(1000);
706 DECLARE sqlsel2 varchar(1000);
707 DECLARE Source varchar(50);
708 /*DECLARE K int DEFAULT 2;
709 DECLARE Pare int ;
710 DECLARE Seq int;*/
711
712 DECLARE csr_user0 CURSOR FOR select youtubelinks_degree.userid from `
        youtubelinks_degree` where youtubelinks_degree.degree = K order by
         userid desc;
713 DECLARE csr_user1 CURSOR FOR select youtubelinks.friendid from `
        youtubelinks` where youtubelinks.userid=PareUser order by
        youtubelinks.friendid;
714 DECLARE csr_user2 CURSOR FOR select youtubelinks.friendid from `
        youtubelinks` where youtubelinks.userid=N1 order by youtubelinks.
        friendid;
715 DECLARE CONTINUE HANDLER FOR NOT FOUND SET rowstatus=1;
716
717
718
719 set tbl = 'youtubesample';
720 set Source = 'youtube';
721
722 /*
723 set Seq = 11;
724 set Pare =1;*/
725
726
727 set @sqldel = concat('delete from ', tbl, ' where ', tbl, '.seq = ',
        Seq);
728 PREPARE stmt FROM @sqldel ;
729 EXECUTE stmt;
730 DEALLOCATE PREPARE stmt;
731
732 OPEN csr_user0;
733 FETCH csr_user0 INTO PareUser;
734 WHILE (rowstatus=0 and num <= Pare) DO
735         select PareUser;
736
737         set @sqlsel0 = concat('select count(*) into @Exsist0 from ',
                tbl, ' where ',tbl, '.userid =', PareUser, ' and ', tbl,
                '.seq = ', Seq);
738         PREPARE stmt FROM @sqlsel0 ;
739         EXECUTE stmt;
740         DEALLOCATE PREPARE stmt;
741
742         if @Exsist0=0 then
743                 set num = num + 1;
744
```

```
745                 set @sqlinsert0 = concat('insert into ', tbl, '  (seq,
                         source, userid) values ( ',  Seq, '\, \'', Source
                        , '\'\,', PareUser, ')');
746                 select @sqlinsert0;
747                 PREPARE stmt FROM @sqlinsert0 ;
748                 EXECUTE stmt;
749                 DEALLOCATE PREPARE stmt;
750         end if;
751         set @Exsist0 = 0;
752
753         OPEN csr_user1;
754         FETCH csr_user1 INTO N1;
755         WHILE rowstatus=0 DO
756
757                 set @sqlsel1 = concat('select count(*) into @Exsist1
                        from ', tbl, ' where ',tbl, '.userid =', N1, ' and
                         ', tbl, '.seq = ', Seq);
758                 PREPARE stmt FROM @sqlsel1 ;
759                 EXECUTE stmt;
760                 DEALLOCATE PREPARE stmt;
761
762                 if N1 = 180 then
763                         select @sqlsel;
764                         select N1;
765                         select @Exsist1;
766                 end if;
767
768                 if @Exsist1=0 then
769                         set @sqlinsert1 = concat('insert into ', tbl,
                                ',  (seq, source, userid) values (', Seq,
                                '\, \'', Source, '\'\,', N1, ')');
770                         PREPARE stmt FROM @sqlinsert1 ;
771                         EXECUTE stmt;
772                         DEALLOCATE PREPARE stmt;
773                 end if;
774                 set @Exsist1 = 0;
775
776                 OPEN csr_user2;
777                 FETCH csr_user2 INTO N2;
778                 WHILE rowstatus=0 DO
779
780                         set @sqlsel2 = concat('select count(*) into
                                @Exsist2 from ', tbl, ' where ', tbl, '.
                                userid =', N2, ' and ', tbl, '.seq = ',
                                Seq);
781                         PREPARE stmt FROM @sqlsel2 ;
782                         EXECUTE stmt;
783                         DEALLOCATE PREPARE stmt;
784
785                         if N2 = 1 then
786                                 select @sqlsel2;
787                                 select N2;
788                                 select @Exsist2;
```

```
789                         end if;
790
791                         if @Exsist2=0 then
792                                 set @sqlinsert2 =concat('insert into
                                        ', tbl, '  (seq, source, userid)
                                        values (', Seq, '\, \'', Source,
                                        '\'\,', N2, ')');
793                                 PREPARE stmt FROM @sqlinsert2 ;
794                                 EXECUTE stmt;
795                                 DEALLOCATE PREPARE stmt;
796                         end if;
797                         set @Exsist2 = 0;
798
799                         FETCH csr_user2 INTO N2;
800                 END WHILE;
801                 CLOSE csr_user2;
802                 set rowstatus =0 ;
803
804                 FETCH csr_user1 INTO N1;
805         END WHILE;
806         CLOSE csr_user1;
807         set rowstatus =0 ;
808
809         FETCH csr_user0 INTO PareUser;
810 END WHILE;
811 CLOSE csr_user0;
812 commit;
813
814 END;
815
816
817
818 -- ----------------------------
819 -- procedure structure for proc_youtubeSampleOrder
820 -- update youtube sample user label , update table youtubesample
821 -- ----------------------------
822 CREATE DEFINER = `amandatian`@`%` PROCEDURE `proc_youtubeSampleOrder`(
        in Seq int)
823 BEGIN
824 /*update youtube sample user label , update table youtubesample*/
825 DECLARE rowstatus int DEFAULT 0;
826 DECLARE User int;
827 DECLARE Num int;
828 DECLARE tbl varchar(40);
829 /*DECLARE Seq int;*/
830 DECLARE csr_user CURSOR FOR select youtubesample.userid from
        youtubesample where youtubesample.seq = Seq order by youtubesample
        .userid  ;
831 DECLARE CONTINUE HANDLER FOR NOT FOUND SET rowstatus=1;
832 SET Num=0;
833 /*SET Seq=11;*/
834 OPEN csr_user ;
835 FETCH csr_user INTO User;
```

```
836 while rowstatus=0 DO
837         SET Num = Num+1;
838         if User = 1156640 THEN
839         select Num;
840         end if;
841         UPDATE `youtubesample` SET youtubesample.userlabel = Num WHERE
                youtubesample.userid = User and youtubesample.seq = Seq;
842          FETCH csr_user INTO User;
843
844 END WHILE;
845 CLOSE csr_user;
846  select Num;
847 END;
848
849 -- ----------------------------
850 -- procedure structure for proc_youtubesamplelinks
851 -- select youtube samples links from youtubelinks by youtube sample
        user, update table youtubesample_links
852 -- ----------------------------
853 CREATE DEFINER = `amandatian`@`%` PROCEDURE `proc_youtubesamplelinks`(
        in Seq int)
854 BEGIN
855 /*select youtube samples links from youtubelinks by youtube sample
        user, update table youtubesample_links */
856 Declare Str varchar(1000000) ;
857 Declare S varchar(1000000) ;
858 Declare UserSample int ;
859 Declare User varchar(20) ;
860 Declare K int;
861 Declare rowstatus int default 0;
862 /*Declare Seq int;*/
863 declare num int default 0;
864
865
866 DECLARE csr_user CURSOR FOR select userid from youtubesample where
        youtubesample.seq = Seq order by youtubesample.userid;
867 DECLARE CONTINUE HANDLER FOR NOT FOUND SET rowstatus=1;
868
869 /*Set Seq = 11;*/
870
871 OPEN csr_user;
872 FETCH csr_user INTO UserSample;
873 Set Str = '(' ;
874 while rowstatus=0 DO
875         set User = concat(UserSample);
876         Set Str = concat(Str, User, ', ');
877         set num = num + 1 ;
878         FETCH csr_user INTO UserSample;
879 END WHILE;
880 CLOSE csr_user;
881 select num;
882
883
```

```
884 set K = length(Str);
885 set Str = substr(Str,1,K-2);
886 set Str = concat(Str, ')');
887 set @Str= concat('INSERT INTO  youtubesample_links ( seq, userid,
        friendid ) SELECT ', Seq, ' , userid, friendid FROM youtubelinks
        where youtubelinks.userid in ', Str);
888
889 delete from youtubesample_links where youtubesample_links.seq = Seq;
890 select @Str;
891
892 PREPARE stmt FROM @Str;
893 EXECUTE stmt;
894 DEALLOCATE PREPARE stmt;
895
896 /*select * from youtubesample_links;*/
897
898 delete from `youtubesample_links`
899 where youtubesample_links.seq = Seq and  youtubesample_links.friendid
        not in (select youtubesample.userid from `youtubesample` where
        youtubesample.seq = Seq);
900
901
902 update youtubesample_links ,youtubesample
903 set youtubesample_links.userlabel = youtubesample.userlabel
904 where youtubesample_links.userid = youtubesample.userid
905 and  youtubesample.seq = Seq
906 and youtubesample_links.seq = Seq;
907
908
909 update `youtubesample_links` ,youtubesample
910 set youtubesample_links.friendlabel = `youtubesample`.userlabel
911 where youtubesample_links.friendid = youtubesample.userid
912 and  youtubesample.seq = Seq
913 and youtubesample_links.seq = Seq;
914
915
916 END;
917
918 -- ----------------------------
919 -- procedure structure for proc_youtubesampleU
920 -- calculate youtube sample users' indegree and outdegree , if
        outdegree is 0 , then set u=1 which is for NL gap, update table
        youtubesample
921 -- ----------------------------
922 CREATE DEFINER = `amandatian`@`%` PROCEDURE `proc_youtubesampleU`(in
        Seq int)
923 BEGIN
924 /*calculate youtube sample users' indegree and outdegree , if
        outdegree is 0 , then set u=1 which is for NL gap, update table
        youtubesample*/
925
926 DECLARE rowstatus int default 0;
927 DECLARE User int;
```

```
928 DECLARE Outdegree int;
929 DECLARE Indegree int;
930 /*DECLARE Seq int;*/
931 DECLARE num int default 0;
932
933 DECLARE csr_user CURSOR FOR select youtubesample.userid from
        youtubesample where youtubesample.seq =Seq ;
934 DECLARE CONTINUE HANDLER FOR NOT FOUND SET rowstatus=1;
935 /*set Seq = 9;*/
936
937 OPEN csr_user;
938 FETCH csr_user INTO User;
939 WHILE  rowstatus=0  DO
940          select count(*) into Outdegree from youtubesample_links where
                 youtubesample_links.userid = User and youtubesample_links.
                 seq =Seq;
941          select count(*) into Indegree from youtubesample_links where
                 youtubesample_links.friendid = User and
                 youtubesample_links.seq =Seq;
942          set num = num +1;
943
944          update youtubesample set youtubesample.outdegree = Outdegree ,
                  youtubesample.indegree = Indegree where youtubesample.
                 userid = User and  youtubesample.seq = Seq;
945          if Outdegree = 0 then
946                  update youtubesample set youtubesample.u=1 where
                         youtubesample.userid = User and youtubesample.seq
                         = Seq;
947          end if;
948          FETCH csr_user INTO User;
949 END WHILE;
950 CLOSE csr_user;
951 commit;
952 /*select num;*/
953
954
955 END;
956
957 -- ----------------------------
958 -- procedure structure for proc_youtubesampleN
959 -- generate youtubesample neighbour list, used to calculate diameter,
        update table youtubesmapleN
960 -- ----------------------------
961 CREATE DEFINER = `amandatian`@`%` PROCEDURE `proc_youtubesampleN`()
962 BEGIN
963 /*generate youtubesample neighbour list, used to calculate diameter,
        update table youtubesmapleN*/
964
965 DECLARE num int default 0;
966 DECLARE N int;
967 DECLARE NPare int;
968 DECLARE insertCnt int;
969 DECLARE strLength int;
```

```
970  DECLARE Nparelist varchar(10000);
971  DECLARE Exsist1 int default 0;
972  DECLARE Exsist2 int default 0;
973  DECLARE PareUser int DEFAULT 1;
974  DECLARE rowstatus int default 0;
975  DECLARE GoStatus int default 0;
976  DECLARE tbl varchar(40);
977  DECLARE sqldel varchar(1000);
978  DECLARE sqlinsert1 varchar(1000);
979  DECLARE sqlsel1 varchar(1000);
980  DECLARE sqlsel2 varchar(1000);
981
982  DECLARE Seq int;
983
984  DECLARE csr_user0 CURSOR FOR select youtubesample.userid from
         youtubesample where youtubesample.seq = Seq  order by
         youtubesample.userid ;
985  DECLARE csr_user1 CURSOR FOR select distinct(youtubesample_links.
         friendid) from youtubesample_links where youtubesample_links.seq =
          Seq and FIND_IN_SET(youtubesample_links.userid, Nparelist);
986  DECLARE csr_user2 CURSOR FOR select distinct(youtubesampleN.friendid)
         from youtubesampleN where youtubesampleN.seq = Seq and
         youtubesampleN.userid=PareUser and youtubesampleN.N=num;
987  DECLARE CONTINUE HANDLER FOR NOT FOUND SET rowstatus=1;
988
989  set tbl = 'youtubesampleN';
990  set Seq = 1;
991
992
993  set @sqldel = concat('delete from ', tbl, ' where ', tbl, '.seq = ',
         Seq);
994  PREPARE stmt FROM @sqldel ;
995  EXECUTE stmt;
996  DEALLOCATE PREPARE stmt;
997
998  OPEN csr_user0;
999  FETCH csr_user0 INTO PareUser;
1000 WHILE rowstatus=0 DO
1001         set num = 0;
1002         insert into youtubesampleN (seq, userid, N, friendid) values (
                 Seq, PareUser, num, Pareuser);
1003         set Nparelist = PareUser;
1004
1005         set GoStatus = 0;
1006
1007         While GoStatus = 0 Do
1008                 set num = num +1;
1009
1010                 set insertCnt = 0;
1011                 OPEN csr_user1;
1012                 FETCH csr_user1 INTO N;
1013                 WHILE rowstatus=0 DO
1014
```

```
1015                        set @sqlsel1 = concat('select count(*) into
                                @Exsist1 from ', tbl, ' where ',tbl, '.
                                friendid =', N, ' and ', tbl, '.userid =
                                ', PareUser,  ' and ', tbl, '.seq = ', Seq
                                );
1016                        PREPARE stmt FROM @sqlsel1 ;
1017                        EXECUTE stmt;
1018                        DEALLOCATE PREPARE stmt;
1019
1020                        if @Exsist1=0 then
1021                                set @sqlinsert1 = concat('insert into
                                        ', tbl, '  (seq,  userid, N,
                                        friendid) values (', Seq, ', ',
                                        PareUser, ', ' , num, ', ',N,  ')
                                        ');
1022                                PREPARE stmt FROM @sqlinsert1 ;
1023                                EXECUTE stmt;
1024                                DEALLOCATE PREPARE stmt;
1025                                set insertCnt = insertCnt +1;
1026                        end if;
1027
1028                        set @Exsist1 = 0;
1029                        FETCH csr_user1 INTO N;
1030                END WHILE;
1031
1032                if insertCnt = 0 then
1033                        set GoStatus = 1;
1034                end if;
1035                CLOSE csr_user1;
1036                set rowstatus =0 ;
1037
1038                set Nparelist = '';
1039                OPEN csr_user2;
1040                FETCH csr_user2 INTO NPare;
1041                WHILE rowstatus=0 DO
1042                        set Nparelist = concat(Nparelist,  Npare, ',')
                                ;
1043                        FETCH csr_user2 INTO NPare;
1044                END WHILE;
1045                CLOSE csr_user2;
1046
1047                set rowstatus =0 ;
1048                set strLength = char_length(Nparelist);
1049                set Nparelist = substring(NpareList, 1, strLength-1);
1050
1051        END WHILE;
1052        FETCH csr_user0 INTO PareUser;
1053 END WHILE;
1054 CLOSE csr_user0;
1055 commit;
1056
1057 /*
1058
```

```
*/


END;



-- ----------------------------
-- procedure structure for proc_youtubesamplecall
-- generate youtubesamples, samples' links, indegree outdegree...
    information in bach (by seq)
-- ----------------------------
CREATE DEFINER = `amandatian`@`%` PROCEDURE `proc_youtubesamplecall`(
    in Seq int, in k int, in Pare int)
BEGIN
/*generate youtubesamples, samples' links, indegree outdegree...
    information in bach (by seq)*/

call proc_youtubesample(Seq, k, Pare);
call proc_youtubesampleorder(Seq);
call proc_youtubesamplelinks(Seq);
call proc_youtubesampleU(Seq);
END;

function [K,A,u,outdegree,indegree,P] = FUNdbA(dbo, account, pswd, tbl
    , seq, sql)
%-----------------output--------------------
% K: amount of nodes
% A: Adjency Matrix
% outdegree: vector of outdegree count
% indegree: vector of indegree count
% P: nomalized adjency matrix
%-------------------------------------------

%-----------------input---------------------
% dbo: dbo name (database server host)
% account: database account
% pswd: database pswd
% tbl: table name
% seq: sample sequence
%-------------------------------------------

% check OS
OS = isunix;
% connect to specified database, specifying username and password
if OS == 1 % unix operation system
    connection = database (dbo, account, pswd,'com.mysql.jdbc.Driver
        ','jdbc:mysql://127.0.0.1:3306/OSN');
else       % windows operation system
    connection = database (dbo, account, pswd);
end

%found how many users are there in samples
```

```
1107
1108  if isempty(sql)
1109      Str1 = ['select u,outdegree, indegree from ', tbl , ' where seq =
          ' , seq, ' order by userlabel'];
1110      Str2 = ['select userlabel,friendlabel from ', tbl, '_links where
          seq = ', seq, ' order by userlabel'];
1111  else
1112      Str1 = sql(1);
1113      Str2 = sql(2);
1114  end
1115
1116  cursor1 = exec(connection, Str1);
1117
1118  % set the reture data format as int/numeric
1119  setdbprefs('DataReturnFormat','numeric');
1120
1121  % retrieve R rows of data
1122  cursor1 = fetch(cursor1);
1123  u=cursor1.data(:,1);
1124  outdegree=cursor1.data(:,2);
1125  indegree=cursor1.data(:,3);
1126  K = length(u);
1127
1128  % open cursor and issue SQL statement to select links
1129  cursor2 = exec(connection, Str2);
1130
1131  % set the reture data format as int/numeric
1132  setdbprefs('DataReturnFormat','numeric');
1133
1134  % retrieve R rows of data
1135  cursor2 = fetch(cursor2);
1136  Data=cursor2.data;
1137  UserLabel= Data(:,1);
1138  FriendLabel = Data(:,2);
1139  Links=length(UserLabel);
1140
1141  A(1:K,1:K)=0;
1142
1143  for i = 1:Links
1144      Row = UserLabel(i);
1145      Col = FriendLabel(i);
1146      A(Row,Col)= 1;
1147  end
1148
1149  P=A;
1150  for i=1:K
1151      if outdegree(i)~=0
1152          for j = 1:K
1153              P(i,j) = A(i,j)/outdegree(i);
1154          end
1155      end
1156  end
1157
```

```
1158
1159 % execute sql to insert data into database
1160 function [Flag] = FUNdbInsert(dbo, account, pswd, tbl, Col, Data, Del)
1161 %-----------------output--------------------
1162 % Flag: nothing
1163 %-------------------------------------------
1164
1165 %-----------------input---------------------
1166 % dbo: dbo name (database server host)
1167 % account: database account
1168 % pswd: database pswd
1169 % tbl: table name
1170 % Col: fields of table
1171 % Data: data to be insert into tbl
1172 %-------------------------------------------
1173 %Col = {'userid','friendid','userlabel','friendlabel','seq','dim','N
        ','a','b'};
1174 %Data = [1,2,1,2,1,1,5,0.7,0.15];
1175 OS = isunix;
1176 % connect to specified database, specifying username and password
1177 if OS == 1 % unix operation system
1178     connection = database (dbo, account, pswd,'com.mysql.jdbc.Driver
            ','jdbc:mysql://127.0.0.1:3306/OSN');
1179 else       % windows operation system
1180     connection = database (dbo, account, pswd);
1181 end
1182
1183 cur = exec(connection, Del);
1184 insert(connection, tbl , Col, Data);
1185 Flag = exec(connection,'commit');
1186
1187
1188 function [Flag] = FUNdegdistFile(FileName,FilePath,DimensionS,
        DimensionE)
1189 % write matrix to file
1190 %-----------------output--------------------
1191 % Flag: nothing, no useful
1192 %-------------------------------------------
1193
1194 %-----------------input---------------------
1195 % A: Data matrix
1196 % FileName: file name
1197 %-------------------------------------------
1198
1199
1200 for i = DimensionS:DimensionE
1201     File = [FilePath FileName];
1202     fid = fopen(File,'r');
1203     if fid == -1
1204         display(['Error opening the ' File]) ;
1205     end
1206     Num=0;
1207     s = sprintf('DegDist%d=[];',i);
```

```
1208      eval(s);
1209      s = sprintf('DegDist%d(Num,:)=Line'';',i);
1210      while feof(fid) == false
1211          Num = Num+1;
1212          Line = fgetl(fid);
1213          Line = sscanf(Line,' %f %f ');
1214          eval(s);
1215      end
1216  end
1217
1218
1219
1220  % plot degree distribution for every dimension
1221  % y:yellow, m:pink, c:light blue, r:red, g:green, b:blue, k:black
1222  Color=['y','m','c','r','g','b','k'];
1223  LegendStr = ['legend(''wiki indegree'',''wiki outdegree'',''wiki
        alldegree''')];
1224  for i = DimensionS:DimensionE
1225      Str = ['plot(log(DegDist%d(:,1)), log(DegDist%d(:,2)),',''')-',
            Color(i),''')'];
1226      s = sprintf(Str,i,i);
1227      eval(s);
1228      xlabel('log(Degree)');
1229      ylabel('log(Distribution)');
1230      LegendStr = [LegendStr ','''GEOP ' num2str(i) ' dimension'''];
1231  end
1232  hold off;
1233  LegendStr = [LegendStr, ')'];
1234  eval(LegendStr);
1235
1236  fileNM = ['GEOP_DegDist_' num2str(N) '_' num2str(DimensionS) '_'
        num2str(DimensionE) ];
1237  saveas(gcf,fileNM);
1238
1239
1240
1241  function [Flag] = FUNfileA(A,FileName,type)
1242  % write matrix to file
1243
1244  %-----------------output--------------------
1245  % Flag: nothing, no useful
1246  %-------------------------------------------
1247
1248  %-----------------input---------------------
1249  % A: Data matrix
1250  % FileName: file name
1251  % type: number type
1252  %       'int': integer
1253  %       'dec': decimal or float
1254  %-------------------------------------------
1255
1256  % A = [1,2,3;4,5,6];
1257
```

```
fid = fopen(FileName,'w');
for i = 1:size(A,1)
    for j = 1:size(A,2)
        if type == 'int'
            fprintf(fid,'%d \t',A(i,j));
        elseif type == 'dec'
            fprintf(fid,'%f \t',A(i,j));
        end
    end
    fprintf(fid,'\r\n', 'end');
end
Flag = fclose(fid);


% verify if a graph is GEN by compare if log(A) and log(Scodd) are
    linear
function [LogA, SCodd] = FUNgen(Lambda, Evector)
%-----------------output--------------------
% LogA: a parameter from the formular
% SCodd: odd walk number begin from nodes
%-------------------------------------------

%-----------------input---------------------
% Lambda: dominant eigenvalue
% Evector: dominant eigenvector
%-------------------------------------------

LogA=log((sinh(Lambda))^(-0.5));

SCodd = (Evector.^2)*sinh(Lambda);

subplot(1,2,1);
plot(log(Evector), log(SCodd));
title('log-log plot');

I = 1:length(Evector);

subplot(1,2,2);
hold on;
plot(I,log(Evector),'-r');
plot(I,LogA + 0.5*log(SCodd),':b');
hold off;
title('right-left match');



function [G] = FUNgeo(N,a,b,m)
% generate cordinators matrix for geop models
%-----------------output--------------------
% G: cordinators of random nodes.
%    G(i,1:m): cordinators
%    G(i,m+1): rank
```

```
%    G(i,m+2): influcial area
%    G(i,m+3): time T of birth
%    G(i,m+4): degree at time T
%    G(i,m+5): rank at time T-1
%    G(i,m+6): if node's influcial area need to be re-evaluated
%               1: yes, rank changed, need to be evaluated
%               0: no, no changes of rank, not need to be evalueated
%-----------------------------------------

%-----------------input--------------------
% N: amount of nodes
% a,b: parameters of GEO-P model, used to calculate influcial area
% m: dimension
%-----------------------------------------


% N=5;
% a = 0.7;
% b=0.15;
% m=2;

if b >= 1-a
    display('wrong parameter b');
end

G = [];
for i=1:N
    G(i,:) = rand(1,m);
end

% rank of every node
%G(:,m+1) = randperm(N);
[ignore,Perm] = sort(rand(1,N));
G(:,m+1) = Perm;

%influcial area of every node
G(:,m+2) = (G(:,m+1).^(-a)).*(N^(-b));

% time T of birth
G(:,m+3)=0;
% initial degree of nodes at time T to be 0
G(:,m+4)=0;
% initial rank of nodes at time T-1 to be the same of at time T
G(:,m+5)=G(:,m+1);
% initial all nodes' degree need to be evalueated.
G(:,m+6)=1;


% resort degree/rank of GEOP datas
function [NewA,NewG] = FUNgeoDeg(A,G,u,a,m)

%-----------------output--------------------
% A: adjency matrix of GEO-P
```

```
%-----------------------------------------

%-----------------input---------------------
% G: cordinates matrix of N nodes
%    G(i,1:m): cordinators
%    G(i,m+1): rank
%    G(i,m+2): influcial area,
%    G(i,m+3): time T of birth
%    G(i,m+4): degree at time T
% u: vectors of m-spare, generated by {-1,0,1}
% m: dimension
%-----------------------------------------

N=size(G,1);
for i = 1:N-1
    d=[];
    % calculate distance of every pair of nodes
    for k = 1:length(u)
        d(k) = norm(G(i,1:m)-G(N,1:m)+ u(k,:),inf);
    end
    D = min(d);

    % check node if node N falls in influtial area of node i
    if m == 2
        Cn = pi;
    else
        if mod(m,2)==1 %odd
            n = (m+1)/2;
            Cn = (2^((m+1)/2))*(pi^((m-1)/2))/((factorial(2*n))/(
                factorial(n)*2^n));
        else           %even
            Cn = (pi^(m/2))/(factorial(m/2));
        end
    end
    r = max((G(i,m+2)/Cn)^(1/m),(G(N,m+2)/Cn)^(1/m));

    %compare distance of two nodes and maxmum influcial raius to
        decide the edge
    if r >= D  % should add an edge
        G(i,m+4) = G(i,m+4)+1;
        G(N,m+4) = G(N,m+4)+1;
        A(i,N)=1;
        A(N,i)=1;
    else
        A(i,N)=0;
        A(N,i)=0;
    end
end

% sort nodes' rank by degree, and calcuclate new influcial area

for i = 1:N
    BigerDeg = length(find(G(:,m+4)>G(i,m+4)));
```

```
1414      OlderAge = length(find(G(:,m+4)==G(i,m+4)&G(:,m+3)< G(i,m+3)));
1415      G(i,m+1)= BigerDeg+OlderAge+1;
1416      G(i,m+2)= (G(i,m+1)^(-a))/(N^a);
1417  end
1418
1419  NewA=A;
1420  NewG=G;
1421
1422
1423  function [A,GT] = FUNgeoedge(G,u,m,A,flag,h)
1424  % create edges for graph generated by geop models
1425  %-----------------output--------------------
1426  % A: adjency matrix of GEO-P
1427  %-------------------------------------------
1428
1429  %-----------------input---------------------
1430  % G: initial cordinates matrix of time 0
1431  %    G(i,1:m): cordinators
1432  %    G(i,m+1): rank
1433  %    G(i,m+2): influcial area
1434  %    G(i,m+3): time T of birth
1435  %    G(i,m+4): degree at time T
1436  %    G(i,m+5): rank at time T-1
1437  %    G(i,m+6): if node's influcial area need to be re-evaluated
1438  %               1: yes, rank changed, need to be evaluated
1439  %               0: no, no changes of rank, not need to be evalueated
1440  % u: vectors of m-spare, generated by {-1,0,1}
1441  % m: dimension
1442  % flag: model flag
1443  %       1: GEOP
1444  %       2: Age
1445  %       3: Degree by changing N
1446  %       4: Degree by fix N
1447  %       5: Age Inverse
1448  %       6: Degree fix N by Radom Ranking & Tension Parameter
1449  %       7: GEOP - Tension Parameter
1450  %       8: GEOP - Pr (radius)
1451  %       9: GEOP - Pr (distance & radius)
1452  %       10:GEOP - Pr (distance & radius Pr(distance)+Pr(radius))
1453  % TnsFlag: count tension parameter?
1454  %               1: yes
1455  %               0: no
1456  % h: Tension Parameter, if TnsFlag=0, h's value does not matter, can
          be 0
1457  % P: probability matrix
1458  %-------------------------------------------
1459
1460  N=size(G,1);
1461  d=[];
1462
1463  if m == 2
1464      Cn = pi;
1465  else
```

```
1466      if mod(m,2)==1 %odd
1467          n = (m+1)/2;
1468          Cn = (2^((m+1)/2))*(pi^((m-1)/2))/((factorial(2*n))/(factorial
                  (n)*2^n));
1469      else           %even
1470          Cn = (pi^(m/2))/(factorial(m/2));
1471      end
1472  end
1473
1474  G(:,m+4)=0;   %clear degree column before check edges
1475
1476  %tension parameter of nodes
1477  TEN = zeros(N,1);
1478  TEN(G(:,m+1)~=0,1) = G(G(:,m+1)~=0,m+1).^h;
1479
1480  %calculate radius and diameter of m-sphear
1481  Far = zeros(1,3);
1482  Far(1,:)=1;
1483  GDiam = norm(Far,inf);
1484  %GDiam = m^0.5;
1485  GRad = 0.5*GDiam;
1486
1487  for i = 1:N
1488      for j = i+1:N
1489          if i == 3 && j == 14
1490              ss = 1;
1491          end
1492          % calculate distance of every pair of nodes
1493          Diff = G(i,1:m)-G(j,1:m);
1494          for k = 1:length(u)
1495               d(k)=norm(Diff+ u(k,:),inf);
1496          end
1497          D = min(d);
1498
1499
1500          % calculate max and min influcial radius of two nodes
1501          MaxI = max(G(i,m+2),G(j,m+2));
1502          rMax = (MaxI/Cn)^(1/m);
1503
1504          MinI = min(G(i,m+2),G(j,m+2));
1505          rMin = (MinI/Cn)^(1/m);
1506
1507          if flag == 7 || flag == 6
1508              %calculate modified influcial radius by tension parameter
1509              TENP = (TEN(i)+TEN(j))/2;
1510              r = rMax*TENP;
1511          else
1512              r = rMax; %take maximum radius of two nodes to compare
                      with distance
1513          end
1514
1515          if flag == 8 % GEOP - Pr (radius)
1516
```

```
1517              if rMax+rMin >= D  % two nodes have overlap area
1518                 Pr = rMin/rMax;
1519                 if rand(1)>=Pr
1520                     A(i,j)=1;
1521                 else
1522                     A(i,j)=0;
1523                 end
1524              else
1525                  A(i,j)=0;
1526              end
1527          elseif flag == 9 % GEOP - Pr (distance & radius)
1528              if rMax+rMin >= D  % two nodes have overlap area
1529                  Pr = (m^0.5-D)/(m^0.5 + 1/rMax + 1/rMin);
1530                  if rand(1)>=Pr
1531                      A(i,j)=1;
1532                  else
1533                      A(i,j)=0;
1534                  end
1535              else
1536                  A(i,j)=0;
1537              end
1538          elseif flag == 10 % GEOP - Pr (distance & radius Pr(distance)+
                  Pr(radius))
1539              if rMax+rMin >= D  % two nodes have overlap area
1540                  PrD = (GRad -D)/GRad;
1541                  PrR = (rMax + rMin)/GDiam;
1542                  Pr = (PrD+PrR)/2;
1543                  if rand(1)>=Pr
1544                      A(i,j)=1;
1545                  else
1546                      A(i,j)=0;
1547                  end
1548                else
1549                  A(i,j)=0;
1550              end
1551          else
1552              %compare distance of two nodes and maxmum influcial raius
                      to decide
1553              %the edge
1554              if r >= D
1555                  A(i,j)=1;
1556              else
1557                  A(i,j)=0;
1558              end
1559          end
1560          %symetry the matrix
1561          A(j,i)=A(i,j);
1562
1563      end
1564      %display(num2str(i));
1565  end
1566
1567  %calculate the degree of matrix
```

```
1568  G(:,m+4) = sum(A)';
1569
1570  GT=G;
1571
1572
1573  function [A,GT] = FUNgeoedgeDeg(G,u,m,A)
1574  % create edges for geop(deg) models
1575  %-----------------output--------------------
1576  % A: adjency matrix of GEO-P
1577  %-------------------------------------------
1578
1579  %-----------------input---------------------
1580  % G: initial cordinates matrix of time 0
1581  %    G(i,1:m): cordinators
1582  %    G(i,m+1): rank
1583  %    G(i,m+2): influcial area
1584  %    G(i,m+3): time T of birth
1585  %    G(i,m+4): degree at time T
1586  %    G(i,m+5): rank at time T-1
1587  %    G(i,m+6): if node's influcial area need to be re-evaluated
1588  %               1: yes, rank changed, need to be evaluated
1589  %               0: no, no changes of rank, not need to be evalueated
1590  % u: vectors of m-spare, generated by {-1,0,1}
1591  % m: dimension
1592  %-------------------------------------------
1593
1594  N=size(G,1);
1595  d=[];
1596
1597  if m == 2
1598      Cn = pi;
1599  else
1600      if mod(m,2)==1 %odd
1601          n = (m+1)/2;
1602          Cn = (2^((m+1)/2))*(pi^((m-1)/2))/((factorial(2*n))/(factorial
                  (n)*2^n));
1603      else           %even
1604          Cn = (pi^(m/2))/(factorial(m/2));
1605      end
1606  end
1607
1608
1609  for i = 1:N
1610      if G(i,m+6)==1  % this node's need to be re-evaluated
1611          for j = 1:N
1612              if i~=j && A(i,j)==0;  % no edge between i,j yet
1613                  % calculate distance of every pair of nodes
1614                  Diff = G(i,1:m)-G(j,1:m);
1615                  for k = 1:length(u)
1616                      d(k)=norm(Diff+ u(k,:),inf);
1617                  end
1618                  D = min(d);
1619
```

```
1620                 % calculate max influcial radius of tow nodes
1621                 MaxI = max(G(i,m+2),G(j,m+2));
1622                 r = (MaxI/Cn)^(1/m);
1623
1624                 %compare distance of two nodes and maxmum influcial
                         raius to decide
1625                 %the edge, and change the value degree : G(:,m+4)
1626                 if r >= D
1627                     A(i,j)=1;
1628                     A(j,i)=1;
1629
1630                     % when i,j nodes both have 1 at m+6 col, it may
                             count
1631                     % the same edge twice, so only do once.
1632                     if (G(j,m+6)==1&&i<j) || (G(j,m+6)==0)
1633                         G(i,m+4) = G(i,m+4)+1;
1634                         G(j,m+4) = G(j,m+4)+1;
1635                     end
1636                 end
1637             end
1638         end
1639         %display(num2str(i));
1640     end
1641 end
1642
1643
1644 GT=G;
1645
1646
1647 % generate GEOP adjency matrix at time 0 and time T with N nodes and m
        dimension
1648 function [A,AT,degree] = FUNgeoP(a,b,N,m,T,flag,modelT0,h)
1649
1650 %-----------------output--------------------
1651 % A: adjency matrix of GEO-P
1652 % AT: adjency matrix of GEO-P at time T
1653 % degree: degree of A
1654 %-------------------------------------------
1655
1656 %-----------------input---------------------
1657 % a,b: GEOP parameters
1658 % N: order of graph
1659 % m: dimention of graph
1660 % T: time T
1661 % flag: choose the algrithom of time changing
1662 %       1: general geop model
1663 %       2: Age
1664 %       3: Degree(changing N)(not available now
1665 %       4: Degree(fix N)
1666 %       5: Age Inverse
1667 %       6: Degree(random rank & Tension Parameter)
1668 %       7: GEOP - Tension Parameter
1669 %       8: GEOP - Pr (radius)
```

```
%       9: GEOP - Pr (distance & radius)
%       10:GEOP - Pr (distance & radius Pr(distance)+Pr(radius))
% modelT0: the model to determine edges at time 0
%       1: geop model
%       2: GNP model
% h: Tension Parameter
%-----------------------------------------

%generate geo model cordinates and influcial area of N nodes
% G: initial cordinates matrix of time 0
%    G(i,1:m): cordinators
%    G(i,m+1): rank
%    G(i,m+2): influcial area
%    G(i,m+3): time T of birth
%    G(i,m+4): degree at time T
%    G(i,m+5): rank at time T-1
%    G(i,m+6): if node's influcial area need to be re-evaluated
%               1: yes, rank changed, need to be evaluated
%               0: no, no changes of rank, not need to be evalueated
[G] = FUNgeo(N,a,b,m);

%calculate u vector of m dimension {-1,0,1}
[u] = FUNgeoU(m);

%generage adjency matrix for geo model based on G
A = zeros(N,N);
if flag ==4 || flag ==6
    if modelT0 == 1        %GEOP
          [AT,GT] = FUNgeoedgeDeg(G,u,m,A);
    elseif modelT0 == 2    %GNP
          [AT,GT] = FUNgnp(N,0.5,G,a,b,m);
    else
           display('undefined model at time 0');
           return;
    end
else
    GT=G;
end

for t = 1: T
    if flag ==1 || flag ==7 || flag ==8 || flag ==9 || flag ==10
        [GT] = FUNgeoTime(GT,m,a,b);
    elseif flag ==2 || flag ==5
        [GT] = FUNgeoTimeAge(GT,m,a,b,t,flag);
    elseif flag ==4 || flag == 6
        [AT,GT] = FUNgeoTimeDegFixN(GT,m,a,b,t,u,AT,flag);
    else
        display('undefined algrithom of time ');
    end
    display(['time:' num2str(t)]);
end

% for modles rather than DegFixN, will determine edges at final time
```

```
if flag ~=4 && flag ~= 6
    if flag == 7   % geop-random rank model, add tension parameter
        [AT,GT] = FUNgeoedge(GT,u,m,A,flag,h);
    else
        [AT,GT] = FUNgeoedge(GT,u,m,A,flag,0);
    end
end
degree = GT(:,m+4);

% resort GEOP nodes degree/rank by degree algorithm GEOP adjency
    matrix at
% time T with T+1 nodes (by degree rank method) with m dimension
function [AT,G,degree] = FUNgeoPDeg(a,b,m,T)

%-----------------output--------------------
% A: adjency matrix of GEO-P at time 0
% AT: adjency matrix of GEO-P at time T by Age algorithm
% degree: degree of A
%-------------------------------------------

%-----------------input---------------------
% a,b: GEOP parameters
% N: order of graph, N = T + 1
% m: dimention of graph
% T: time T
%-------------------------------------------


% initial G at time 0 with only one node
G(1,1:m) = rand(1,m);           % cordinators
G(1,m+1)=1;                     % rank
G(1,m+2)=(G(1,m+1)^(-a))/(2^a);        % influcial area
G(1,m+3)=0;                     % time of birth
G(1,m+4)=0;                     % degree of node at time T

A=[0];

%calculate u vector of m dimension {-1,0,1}
[u] = FUNgeoU(m);

for t = 1:T
    G(t+1,1:m)=rand(1,m);
    G(t+1,m+1)= 0;
    G(t+1,m+2)= 0;
    G(t+1,m+3)= t;
    G(t+1,m+4)= 0;
    [NewA,NewG]=FUNgeoDeg(A,G,u,a,m);
    A=NewA;
    G=NewG;
end
AT=A;

degree = G(:,m+4);
```

```
1775
1776
1777
1778
1779  function [GT] = FUNgeoTime(G,m,a,b,T)
1780  % generate cordinators matrix by different time T
1781  %-----------------output--------------------
1782  % GT: cordinate matrix of time T
1783  %-------------------------------------------
1784
1785  %-----------------input---------------------
1786  % G: initial cordinates matrix of time 0
1787  %    G(i,1:m): cordinators
1788  %    G(i,m+1): rank
1789  %    G(i,m+2): influcial area
1790  %    G(i,m+3): time T of birth
1791  %    G(i,m+4): degree at time T
1792  %    G(i,m+5): rank at time T-1
1793  %    G(i,m+6): if node's influcial area need to be re-evaluated
1794  %               1: yes, rank changed, need to be evaluated
1795  %               0: no, no changes of rank, not need to be evalueated
1796  % m: dimension
1797  % a,b: parameters of GEO-P model, used to calculate influcial area
1798  %-------------------------------------------
1799
1800
1801  N=size(G,1);
1802  % generate new cordinagtes
1803  NewCor = rand(1,m);
1804  % generate new rank
1805  NewRank = 1+fix(N*rand(1,1));
1806  %generate new row for GT
1807  New = [NewCor, NewRank,0,T,0,0,1];
1808  GT = [G(:,1:m+1);New];
1809
1810  % re-order existing rank by adding a new node
1811  for i = 1:N
1812      if GT(i,m+1)>= NewRank
1813          GT(i,m+1)=GT(i,m+1)+1;
1814      end
1815  end
1816
1817  % choose and delete a node from existing nodes
1818  DelRank = 1+fix((N+1)*rand(1,1));
1819  GT(GT(:,m+1)==DelRank,:)=[];
1820  % re-order existing ran by deleting a node
1821  for i = 1:N
1822      if GT(i,m+1)>= DelRank
1823          GT(i,m+1)=GT(i,m+1)-1;
1824      end
1825  end
1826
1827  % calculate influcial area
```

```
GT(:,m+2) = (GT(:,m+1).^(-a)).*(N^(-b));


% generate adjency matrix for GEOP-DegFixN model, literated by degeree
    algrithom
function [A,GT] = FUNgeoTimeDegFixN(G,m,a,b,T,u,A,flag)
%-----------------output--------------------
% GT: cordinate matrix of time T
%-------------------------------------------

%-----------------input---------------------
% G: initial cordinates matrix of time 0
%    G(i,1:m): cordinators
%    G(i,m+1): rank
%    G(i,m+2): influcial area
%    G(i,m+3): time T of birth
%    G(i,m+4): degree at time T
%    G(i,m+5): rank at time T-1
%    G(i,m+6): if node's influcial area need to be re-evaluated
%               1: yes, rank changed, need to be evaluated
%               0: no, no changes of rank, not need to be evalueated
% m: dimension
% a,b: parameters of GEO-P model, used to calculate influcial area
% T: time t
% u: vectors of m-dimension (-1,0,1)^m
% A: adjency matrix
% flag: model type
%       1: GEOP
%       2: Age
%       3: Degree by changing N
%       4: Degree by fix N
%       5: Age Inverse
%       6: Degree fix N by Radom Ranking
%       7: GEOP - Random Rank
%       8: GEOP - Pr
%-------------------------------------------


N=size(G,1);

% choose and delete a node from first N existing nodes
DelRow = 1+fix((N)*rand(1,1));
% if there was edge between this node, deleted node,then delete the
    edge
for j=1:N
    if A(DelRow,j)==1
        G(j,m+4)=G(j,m+4)-1;
    end
end
G(DelRow,:)=[];
A(DelRow,:)=[];
A(:,DelRow)=[];

```

```
% generate new cordinagtes
NewCor = rand(1,m);
%generate new row for GT
New = [NewCor, 0,0,T,0,0,1];
G = [G;New];
A(N,:)=0;
A(:,N)=0;

% set value of rank at time T-1;
G(:,m+5)=G(:,m+1);

if flag ==6  %random rank

    e1=0.1;
    e2=-0.1;
    [A,G] = FUNgeoedge(G,u,m,A,flag,0);
    % sort nodes' rank by degree
    E=sum(G(:,m+4))/2;
    MaxDeg = max(G(:,m+4));
    %ExtP = (log(E)/log(MaxDeg))*0.9;
    ExtP = (log(E)/log(MaxDeg))^e1;
    G(:,m+1)=min(((G(:,m+4).^ExtP)./E).^e2,10*N);
else   %consecutive rank

    [A,G] = FUNgeoedgeDeg(G,u,m,A);
    % sort nodes' rank by degree
    for i = 1:N
        BigerDeg = length(find(G(:,m+4)>G(i,m+4)));
        OlderAge = length(find(G(:,m+4)==G(i,m+4)&G(:,m+3)< G(i,m+3)))
            ;
        G(i,m+1)= BigerDeg+OlderAge+1;
    end
end

% calculate influcial area
G(:,m+2) = (G(:,m+1).^(-a)).*(N^(-b));

% rank changed, set the flat to be 1 (need to be evaluated next time)
G(find(G(:,m+1)~=G(:,m+5)),m+6)=1;
% rank stays the same, set the flat to be 0 (no need to be evaluated
    next time)
G(find(G(:,m+1)==G(:,m+5)),m+6)=0;

GT=G;


% gegerate cordinate for GEOP-Age model by time T
function [GT] = FUNgeoTimeAge(G,m,a,b,T,flag)
%-----------------output--------------------
% GT: cordinate matrix of time T
%-------------------------------------------

%-----------------input---------------------
```

```
% G: cordinators of random nodes.
%    G(i,1:m): cordinators
%    G(i,m+1): rank
%    G(i,m+2): influcial area
%    G(i,m+3): time T of birth
%    G(i,m+4): degree at time T
%    G(i,m+5): rank at time T-1
%    G(i,m+6): if node's influcial area need to be re-evaluated
%               1: yes, rank changed, need to be evaluated
%               0: no, no changes of rank, not need to be evalueated
% m: dimension
% a,b: parameters of GEO-P model, used to calculate influcial area
% T: time t
% flag: indicate the model type
%       2. Age
%       5. Age-Inverse
%-----------------------------------------


N=size(G,1);
% generate new cordinagtes
NewCor = rand(1,m);
% generate new rank, the smallest rank
if flag == 2     % by age
    NewRank = N+1;
elseif flag == 5    %by age-inverse model
    NewRank = 1;
else
    display('undefined modle type for age');
    return;
end

%generate new row for GT
New = [NewCor, NewRank,0,T,0,0,0];
GT = [G;New];

% choose and delete a node from existing nodes
DelRow = 1+fix(N*rand(1,1));
DelRank = GT(DelRow,m+1);
GT(DelRow,:)=[];

% set value of rank at time T-1;
G(:,m+5)=G(:,m+1);

% re-order existing rank by deleting a node
for i = 1:N
     if flag == 2    % by age model
        if GT(i,m+1)>= DelRank
            GT(i,m+1)=GT(i,m+1)-1;
        end
     elseif flag == 5   %by age-inverse model
         if i < N
             if GT(i,m+1) < DelRank
```

```
                    GT(i,m+1)=GT(i,m+1)+1;
                end
            end
        end
    end
% calculate influcial area
GT(:,m+2) = (GT(:,m+1).^(-a)).*(N^(-b));


function [u] = FUNgeoU(m)
% generate u vectors in R^m by {-1,0,1}, used to calculate distance of
% veritices
%-----------------output--------------------
% u: vectors of m-spare, generated by {-1,0,1}
%-------------------------------------------

%-----------------input---------------------
% m: dimenstion
%-------------------------------------------


Str1 = '[';
Str2 = ']=ndgrid(';
Str3 = 'u=[';
for i = 1:m
    Str1 = [Str1,'v',num2str(i),','];
    Str2 = [Str2,'-1:1,'];
    Str3 = [Str3,'v',num2str(i),'(:),'];
end

Str1 = Str1(1:length(Str1)-1);

Str2 = Str2(1:length(Str2)-1);
Str2 = [Str2,')'];

Str1 = [Str1, Str2, ';'];

Str3 = Str3(1:length(Str3)-1);
Str3 = [Str3,'];'];

eval(Str1);
eval(Str3);


%generate G(K,1/2) adjency matrix
function [A,GT] = FUNgnp(N,P,G,a,b,m)

%-----------------output--------------------
% A: Adjency Matrix of order K
%-------------------------------------------

%-----------------input---------------------
```

```
% K: order of graph
% P: probability of edge between any pair of nodes
% G: cordinators of random nodes.
%    G(i,1:m): cordinators
%    G(i,m+1): rank
%    G(i,m+2): influcial area
%    G(i,m+3): time T of birth
%    G(i,m+4): degree at time T
%    G(i,m+5): rank at time T-1
%    G(i,m+6): if node's influcial area need to be re-evaluated
%               1: yes, rank changed, need to be evaluated
%               0: no, no changes of rank, not need to be evalueated
%-----------------------------------------

A=[];
A=rand(N,N);
for i=1:N
    A(i,i)=0;
    for j=i+1:N
        if A(i,j)>=P
            A(i,j)=1;
        else
            A(i,j)=0;
        end
        A(j,i)=A(i,j);
    end
    Degree = sum(A(i,:));
    G(i,m+4) = Degree;
end


% sort nodes' rank by degree
for i = 1:N
    BigerDeg = length(find(G(:,m+4)>G(i,m+4)));
    OlderAge = length(find(G(:,m+4)==G(i,m+4)&G(:,m+3)< G(i,m+3)));
    G(i,m+1)= BigerDeg+OlderAge+1;
end

% calculate influcial area
G(:,m+2) = (G(:,m+1).^(-a)).*(N^(-b));

GT=G;



% generate Normalized Loplacion matrix and calculate spectual gap
function [inL,outL,indegree, outdegree,inLambda_N, inLambda_2,
    outLambda_N, outLambda_2,inGap,outGap] = FUNLPgap(A)

%-----------------output--------------------
% A: adjency matrix
%-----------------------------------------
```

```
%-----------------input---------------------
% L: Nomalized Loplcion matrix
% indegree: indegree
% outdegree: outdegree
% Lambda_N: biggest eigenvalue
% Lambda_2: 2nd smalled eigenvalue
% gap: max{abs(Lambda_N - 1),abs(Lambda_2 - 1)}
%-----------------------------------------

N = length(A);

%calculate indegree and out degree
indegree = [];
outdegree = [];
indegree_D = [];
outdegree_D = [];
for i = 1:N
    outdegree(i,1) = sum(A(i,:));
    indegree(i,1) = sum(A(:,i));

    if outdegree(i,1) == 0
        outdegree_D(i,1) = 0;
    else
        outdegree_D(i,1) = outdegree(i,1)^(-0.5);
    end

    if indegree(i,1) == 0
        indegree_D(i,1) = 0;
    else
        indegree_D(i,1) = indegree(i,1)^(-0.5);
    end

end

%generate D matrix with diagonal elements as degree
inD = diag(indegree_D);
outD = diag(outdegree_D);
I = eye(N);

inL = I - inD*A*inD;
outL = I - outD*A*outD;

[inV,inLambda] = eig(inL);
[outV,outLambda] = eig(outL);

inLambda = sort(diag(inLambda));
outLambda = sort(diag(outLambda));

inLambda_N = inLambda(N);
inLambda_2 = inLambda(2);
outLambda_N = outLambda(N);
outLambda_2 = outLambda(2);

```

```
inGap = max(abs(inLambda(N)-1),abs(inLambda(2)-1));
outGap = max(abs(outLambda(N)-1),abs(outLambda(2)-1));


% calculate dominent eigenvalue and eigenvector by powermethod
function [Lambda, V,count] = FUNpowermethod(A, itermax, errmax)
%-----------------output--------------------
% Lambda: dominant eigenvalue
% V: dominant eigenvector
% count: for test
%-------------------------------------------

%-----------------input---------------------
% Lambda: dominant eigenvalue
% Evector: dominant eigenvector
%-------------------------------------------

% itermax=10;
% errmax = 0.001;
% A=[1,2,3;4,1,3;0,3,4];

N = size(A, 1) ;
Lambda = 0 ;
V = ones(N, 1) ;

%calculator dominant eigenvector V
for num = 1 : itermax
    Lambdaold=Lambda;
    V = A * V ;
    [Lambda, i] = max(abs(V)) ;
    V = V./norm(V); % normlize
    errtemp = abs((Lambda-Lambdaold)/Lambda) ; % calculate the error
    if(errtemp < errmax)
        break ;
    end
end

P = A*V;
Lambda=0;
count=0;
% for i = 1:N
%     if V(i)~= 0
%         Lambda=Lambda+ abs(P(i)/V(i));
%         count = count+1;
%     end
%
% end
% Lambda=Lambda/count;

for i = 1:N
    if V(i)~= 0
        Lambda=abs(P(i)/V(i));
        break;
```

```
2194      end
2195
2196  end
2197
2198
2199  % alculate adjency matrix of GEOP and changing by time T by different
2200  % algorithm, save edges into file and data into database
2201  function [] = GEOP_dbinsert(N, T, a, b, Seq, dimensionS, dimensionE,
          flag,modelT0,h)
2202
2203  %-----------------output--------------------
2204  %-------------------------------------------
2205
2206  %-----------------input---------------------
2207  % N: Order of Graph
2208  % T: time T for GT
2209  % a,b: parameters of GEOP model
2210  % Seq: sequence num of samples with same parameters (N,a,b,dim,T)
2211  % dimensionS: dimension start num for loop
2212  % dimensionE: dimension end num for loop
2213  % flag: model type
2214  %       1: GEOP
2215  %       2: Age
2216  %       3: Degree by changing N
2217  %       4: Degree by fix N
2218  %       5: Age Inverse
2219  %       6: Degree fix N by Radom Ranking & Tension Parameter
2220  %       7: GEOP - Tension Parameter
2221  %       8: GEOP - Pr (radius)
2222  %       9: GEOP - Pr (distance & radius )
2223  %       10:GEOP - Pr (distance & radius Pr(distance)+Pr(radius))
2224  % modelT0: the model to determine edges at time 0
2225  %       1: geop model
2226  %       2: GNP model
2227  % h: Tension Parameter, can input 0 if do not need this parameter
2228  %-------------------------------------------
2229
2230  format short e
2231  % N = 7115;
2232  % T = 1;
2233  % a = 0.7;
2234  % b = 0.15;
2235  % Seq = 1;
2236
2237  tbl = 'geopsample_links';
2238  Col = {'userid','friendid','userlabel','friendlabel','seq','dim','N','
          a','b','T' };
2239
2240
2241  for i = dimensionS:dimensionE   %dimension
2242      DA = [];
2243      A=[];
2244      AT=[];
```

```
2245      if flag ==3
2246          [AT,G,degree]=FUNgeoPDeg(a,b,i,T);
2247          N=T+1;
2248      else
2249          [A,AT,degree]=FUNgeoP(a,b,N,i,T,flag,modelTO,h);
2250      end
2251
2252      display('FUNgeoP done');
2253      Del = sprintf('delete from geopsample_links where seq = %d and dim
               = %d and N= %d and a = %d and b = %d and T=%d' , Seq, i,N , a
          , b, T);
2254      Data = zeros(N^2,10);
2255      Edge = zeros(N^2,2);
2256      count = 1;
2257      for m = 1:N
2258          for n = 1:N
2259              if AT(m,n) == 1
2260                  Data(count,:) = [m,n,m,n,Seq,i,N,a,b,T];
2261                  Edge(count,:) = [m,n];
2262                  count = count+1;
2263              end
2264          end
2265          %display(['m=' num2str(m)]);
2266      end
2267      Data=Data(1:count-1,:);
2268      Edge=Edge(1:count-1,:);
2269      display('DataEdge done');
2270      FileName = ['GEOP_dbinsert_' num2str(N) '_' num2str(T) '_' num2str
          (Seq) '_' num2str(100*a) '_' num2str(100*b) '_' num2str(i) '.
          txt'];
2271      [Flag] = FUNfileA(Edge,FileName,'int');
2272      StrDis = sprintf('%d th dimension edge data is saved to file', i);
2273      display(StrDis);
2274
2275      FileNameTB = ['GEOP_dbinsert_' num2str(N) '_' num2str(T) '_'
          num2str(Seq) '_' num2str(100*a) '_' num2str(100*b) '_' num2str
          (i) '_tbl.txt'];
2276      [Flag] = FUNfileA(Data,FileNameTB,'int');
2277      StrDis = sprintf('%d th dimension edge data is saved to table file
          ', i);
2278      display(StrDis);
2279
2280
2281      %[Flag] = FUNdbInsert('OSN', 'root', 'mysql_root', tbl, Col, Data,
               Del);
2282      %StrDis = sprintf('%d th dimension is inserted to db', i);
2283      %display(StrDis);
2284  end
2285
2286
2287  % calculate and save degree distribution list to file
2288  function [] = GEOP_DegDist_db(N, T, a, b, Seq, dimensionS, dimensionE)
2289
```

```
%-----------------output--------------------
%-------------------------------------------

%-----------------input---------------------
% N: Order of Graph
% T: time T for GT
% a,b: parameters of GEOP model
% Seq: sequence num of samples with same parameters (N,a,b,dim,T)
% dimensionS: dimension start num for loop
% dimensionE: dimension end num for loop
%-------------------------------------------
dbo = 'OSN';
account = 'root';
pswd = 'mysql_root';

% Seq = 1;
% T = 0;
% N = 1000;
% a = 0.7;
% b = 0.15;
% DimensionS = 6;
% DimensionE = 6;

Data = [];
DataA = [];
Degree = [];



OS = isunix;
% connect to specified database, specifying username and password
if OS == 1 % unix operation system
    connection = database (dbo, account, pswd,'com.mysql.jdbc.Driver
        ','jdbc:mysql://127.0.0.1:3306/OSN');
else       % windows operation system
    connection = database (dbo, account, pswd);
end

% generate GEOP matrix and calculate degree for every node of GEOP
for i = dimensionS:dimensionE   %dimension
    DegDist = [];

    Str = sprintf('select degree, dist from geopdegdist where seq=%d
        and N = %d and dim =%d and a = %d and b = %d and T=%d', Seq,N,
        i,a,b,T);
    cursor = exec(connection, Str);

    % set the reture data format as int/numeric
    setdbprefs('DataReturnFormat','numeric');

    % retrieve indegree distribution
    cursor = fetch(cursor);
    DegDist(:,1)=cursor.data(:,1);
```

```
2340      DegDist(:,2)=cursor.data(:,2);
2341
2342      FileName = ['GEOP_DegDist_', num2str(N) '_' num2str(T) '_' num2str
              (Seq) '_' num2str(a*100) '_' num2str(b*100) '_' num2str(i) '.
              txt'];
2343      [Flag] = FUNfileA(DegDist,FileName,'int');
2344  end
2345
2346
2347  % read degree distribution file and plot different dim in one graph
2348  function [LegendStr,TitleStr] = GEOP_DegDist_plot(N, T, a, b, Seq,
          dimensionS, dimensionE,FilePath,Flag)
2349
2350  %-----------------output--------------------
2351  %-------------------------------------------
2352
2353  %-----------------input---------------------
2354  % N: Order of Graph
2355  % T: time T for GT
2356  % a,b: parameters of GEOP model
2357  % Seq: sequence num of samples with same parameters (N,a,b,dim,T)
2358  % dimensionS: dimension start num for loop
2359  % dimensionE: dimension end num for loop
2360  % FilePath: the degree distribution list file path,if it is null then
          take
2361  % the default value
2362  % flag: model type
2363  %       1: GEOP
2364  %       2: Age
2365  %       3: Degree by changing N
2366  %       4: Degree by fix N
2367  %       5: Age Inverse
2368  %       6: Degree fix N by Radom Ranking & Tension Parameter
2369  %       7: GEOP - Tension Parameter
2370  %       8: GEOP - Pr (radius)
2371  %       9: GEOP - Pr (distance & radius )
2372  %       10:GEOP - Pr (distance & radius Pr(distance)+Pr(radius))
2373  %-------------------------------------------
2374
2375  % N=1000;
2376  % T=0;
2377  % a = 0.7;
2378  % b = 0.15;
2379  % DimensionS=1;
2380  % DimensionE=6;
2381
2382  hold on;
2383  % Open GEOP_DegDist_%.txt file to plot degree distribution
2384  if isempty(FilePath)
2385     FilePath = 'C:\Amanda\Ryerson\thesis\mfile\main\output\GEOP_DegDist
              \';
2386  end
2387
```

```
% open different dimension file and read the data into vectors
for i = dimensionS:dimensionE
    FileName = ['GEOP_DegDist_' num2str(N) '_' num2str(T)  '_' num2str
        (Seq) '_' num2str(a*100) '_' num2str(b*100) '_' num2str(i) '.
        txt'];
    File = [FilePath FileName];
    fid = fopen(File,'r');
    if fid == -1
        display(['Error opening the ' File]) ;
    end
    Num=0;
    s = sprintf('DegDist%d=[];',i);
    eval(s);
    s = sprintf('DegDist%d(Num,:)=Line'';',i);
    while feof(fid) == false
        Num = Num+1;
        Line = fgetl(fid);
        Line = sscanf(Line,' %f %f ');
        eval(s);
    end
end



% plot degree distribution for every dimension
% y:yellow, m:pink, c:light blue, r:red, g:green, b:blue, k:black
Color=['m','r','k','b','g','y','c'];
Line = {'-', '--', ':', '-.','-+','-h'};
LegendStr = [''];
for i = dimensionS:dimensionE
    Linestyle = Line{i};
    Str = ['plot(log(DegDist%d(:,1)), log(DegDist%d(:,2)),','''',
        Linestyle,Color(i),''')'];
    s = sprintf(Str,i,i);
    eval(s);
    xlabel('log(Degree)');
    ylabel('log(Degree Distribution)');
    LegendStr = [LegendStr ','''GEO-P ' num2str(i) ' dimension'''];
end
% list models to display on graphs
Model = {'GEO-P','GEO-P(Age)','GEO-P(Deg)','GEO-P(GnpDeg)','GEO-P(Ten)
    ','GEO-P(DegRdmRk)'};

LegendStr = substring(LegendStr, 1, length(LegendStr)-1);
LegendStr = ['legend(', LegendStr, ')'];
eval(LegendStr);
TitleStr = ['model:' Model{Flag} '   Seq:' num2str(Seq) '   N:'
    num2str(N) '   T:' num2str(T) '   a:' num2str(a) '   b:' num2str(b
    ) '   p:1 '];
title(TitleStr);

```

```
2433  fileNM = ['GEOP_DegDist_' num2str(N) '_' num2str(T) '_' num2str(Seq) '
          _' num2str(a*100) '_' num2str(b*100) '_' num2str(dimensionS) '_'
          num2str(dimensionE) ];
2434  saveas(gcf,fileNM);
2435
2436
2437  % alculate spectral gap of GEOP and compare with GNP(random graph)
2438  function [] = GEOP_gap_db(N, T, a, b, Seq, dimensionS, dimensionE,flag
          )
2439
2440  %-----------------output--------------------
2441  %-------------------------------------------
2442
2443  %-----------------input---------------------
2444  % N: Order of Graph
2445  % T: time T for GT
2446  % a,b: parameters of GEOP model
2447  % Seq: sequence num of samples with same parameters (N,a,b,dim,T)
2448  % dimensionS: dimension start num for loop
2449  % dimensionE: dimension end num for loop
2450  % flag: compare with GNP?
2451  %       0:no
2452  %       1:yes
2453  %-------------------------------------------
2454
2455  % Seq = 1;
2456  % N = 7115;
2457  % T = 0;
2458  % a = 0.7;
2459  % b = 0.15;
2460  DataA = [];
2461  Data = [];
2462  for i = dimensionS:dimensionE   %dimension
2463      A = [];
2464      DA = [];
2465      sql1 = sprintf('select u,outdegree, indegree from geopsample where
               seq = %d and N= %d and dim = %d and a = %d and b = %d and T=%
              d order by userlabel', Seq,N,i,a,b,T);
2466      sql2 = sprintf('select userlabel,friendlabel from geopsample_links
               where seq = %d and N= %d and dim = %d and a = %d and b = %d
              and T=%d order by userlabel', Seq,N,i,a,b,T);
2467      sql ={sql1;sql2};
2468      [K,A] = FUNdbA('OSN', 'root', 'mysql_root', 'geopsample','1',sql);
2469      N = length(A);
2470
2471      % calculate 1st largest eigenvalue
2472      [Lambda,Evector]=FUNpowermethod(A,20,0.001);
2473
2474      % calculate 2nd largest eigenvalue
2475      A1=A-Lambda*Evector*Evector';
2476      [Lambda2,Evector2]=FUNpowermethod(A1,20,0.001);
2477      Gap = Lambda - Lambda2;
2478
```

```
2479      D = {'GEOP',num2str(N),num2str(Seq),num2str(i),num2str(Lambda),
              num2str(Lambda2),num2str(Gap)};
2480      Data = [Data;D];
2481      DA = [0,N,Seq,i,Lambda,Lambda2,Gap];
2482      DataA = [DataA;DA];
2483
2484
2485  end
2486
2487  % GNP
2488  if flag ==1
2489      [G]=FUNgnp(N,1/2);
2490      [GNPLambda,GNPEvector]=FUNpowermethod(G,200,0.001);
2491
2492      % calculate 2nd largest eigenvalue
2493      G1=G-GNPLambda*GNPEvector*GNPEvector';
2494      [GNPLambda2,GNPEvector2]=FUNpowermethod(G1,200,0.001);
2495
2496      GNPGap = GNPLambda - GNPLambda2;
2497      GNPD = {'GNP', num2str(N),'1','1',num2str(GNPLambda),num2str(
              GNPLambda2),num2str(GNPGap)};
2498      Data = [Data;GNPD];
2499      DataA = [DataA;1,N,1,1,GNPLambda,GNPLambda2,GNPGap];
2500  end
2501
2502  filename = ['GEOP_gap_' num2str(N) '_' num2str(T) '_' num2str(Seq) '_'
          num2str(100*a) '_' num2str(100*b)];
2503  filenametxt = [filename '.txt'];
2504  [Flag] = FUNfileA(DataA,filenametxt, 'dec');
2505
2506
2507  f = figure('Position',[10 10 700 700]);
2508  cnames = {'model','nodes','Seq', 'm-dimension','Lambda_1','Lambda_2','
         Gap',};
2509  t = uitable('Data',Data,'ColumnName',cnames,'parent',f,'Position',[1 1
          700 700]);
2510  saveas(gcf,filename);
2511
2512  % plot the graph simulate by GEO-P model
2513  function [G,AT] = GEOP_sim(N,m,a,b,File,WF,circle,edge)
2514
2515  %-----------------output--------------------
2516  %-------------------------------------------
2517
2518  %-----------------input---------------------
2519  % N: Order of Graph
2520  % m: dimension
2521  % a,b: GEOP model parameters
2522  % File: file name of vertices information, format is same to G
2523  % Wf: if write the G to file
2524  %     1: yes
2525  %     0: no
2526  % circle: which vertices will be plot influence region
```

```
% edge: if plot edges between vertices
%     1: yes
%     0: no
%-----------------------------------------

%-----------------output--------------------
% G: cordinators of random nodes.
%    G(i,1:m): cordinators
%    G(i,m+1): rank
%    G(i,m+2): influcial area
%    G(i,m+3): time T of birth
%    G(i,m+4): degree at time T
%    G(i,m+5): rank at time T-1
%    G(i,m+6): if node's influcial area need to be re-evaluated
%               1: yes, rank changed, need to be evaluated
%               0: no, no changes of rank, not need to be evalueated
%-----------------------------------------
if isempty(File)
    [G] = FUNgeo(N,a,b,m);
else
    fid = fopen(File,'r');
    if fid == -1
        display(['Error opening the ' File]) ;
    end
    row=0;
    G = zeros(N,m+6);
    while feof(fid) == false
        row = row+1;
        Line = fgetl(fid);
        Line = sscanf(Line,' %f %f %f %f %f %f %f %f ');
        G(row,:)=Line;
    end
end

[u] = FUNgeoU(m);
A = zeros(N,N);
[AT,GT] = FUNgeoedge(G,u,m,A,1,0);

if m == 2
    Cn = pi;
else
    if mod(m,2)==1 %odd
        n = (m+1)/2;
        Cn = (2^((m+1)/2))*(pi^((m-1)/2))/((factorial(2*n))/(factorial
            (n)*2^n));
    else           %even
        Cn = (pi^(m/2))/(factorial(m/2));
    end
end
hold on;
axis([0 1 0 1]);
axis square;
seta=0:0.001:2*pi;
```

```
Data = zeros(N,m+3);
for i = 1:N
    plot(G(i,1),G(i,2),'.r','Markersize',15);
    rank = G(i,m+1);
    text(G(i,1),G(i,2)-0.03,num2str(rank),'FontSize',8);
end

% plot circle
for i = 1:length(circle)
    v = circle(i);
    % calculate max and min influcial radius of two nodes
    I = G(G(:,m+1)==v,m+2);
    r = (I/Cn)^(1/m);

% give smaller radius for plot effections only
%     if v==7
%         r = 0.19;
%     elseif v==16
%         r=0.13;
%     end

    x=G(G(:,m+1)==v,1)+r*cos(seta);
    y=G(G(:,m+1)==v,2)+r*sin(seta);
    plot(x,y);
end

if edge == 1
    for i = 1:N
        for j = i+1:N
            if AT(i,j) == 1
                irank = G(i,m+1);
                jrank = G(j,m+1);
                plot([G(i,1) G(j,1)],[G(i,2) G(j,2)]);
            end
        end
    end
end


hold off;

if WF == 1
    FileName = ['GEOP_sim_' num2str(N) '_' num2str(m) '_' num2str(100*
        a) '_' num2str(100*b) '.txt'];
    [Flag] = FUNfileA(G,FileName,'dec');
end



% generate G(n,p) edges file
function [] = GNP_edge(N, P)

%-----------------output--------------------
```

```
%-----------------------------------------

%-----------------input---------------------
% N: Order of Graph
% P: probability of edeges
%-----------------------------------------

[A,GT]=FUNgnp(N,P,0,0,0,0);
Data = zeros(N,2);
count = 0;
for i=1:N
   for j=i+1:N
        if A(i,j)==1
            count = count + 1;
            A(i,j)=1;
            Data(count,1) = i;
            Data(count,2) = j;
        end
   end
end

FileName = ['GNP_' num2str(N) '_' num2str(P*100) '_edge.txt'];

[Flag] = FUNfileA(Data,FileName,'int');



% generate G(n,p) model and calculate the spetural gap
function [] = Gnp_gap(N, P)

%-----------------output--------------------
%-----------------------------------------

%-----------------input---------------------
% N: Order of Graph
% P: probability of edeges
%-----------------------------------------

[A,GT]=FUNgnp(N,P,0,0,0,0);

[Lambda,Evector]=FUNpowermethod(A,200,0.00001);

% calculate 2nd largest eigenvalue
A1=A-Lambda*Evector*Evector';
[Lambda2,Evector2]=FUNpowermethod(A1,200,0.00001);
%[Lambda,Lambda2,Evector] = eigen(G);
Gap = Lambda - Lambda2;

Data = [Lambda, Lambda2, Gap];
FileName = ['GNP_' num2str(N) '_' num2str(P*100) '.txt'];

[Flag] = FUNfileA(Data,FileName,'dec');

```

```
2684
2685 % calculate spectral gap for wiki, and verify if it satisfy gen graph
2686 function [] = wiki_gap_gen()
2687
2688 %-----------------output--------------------
2689 %-------------------------------------------
2690
2691 %-----------------input---------------------
2692 %-------------------------------------------
2693 Data = [];
2694 [K,A] = FUNdbA('OSN', 'root', 'mysql_root', 'wikisample','0',[]);
2695 N = length(A);
2696
2697 % calculate dominent eigenvector by power method
2698 [Lambda,Evector]=FUNpowermethod(A,50,0.000001);
2699
2700 % calculate 2nd largest eigenvalue
2701 A1=A-Lambda*Evector*Evector';
2702 [Lambda2,Evector2]=FUNpowermethod(A1,51,0.00001);
2703
2704 %[Lambda,Lambda2,Evector] = eigen(A);
2705
2706 % plot and save gap
2707 Gap = Lambda - Lambda2;
2708 Data = {'wiki', '0', num2str(N), num2str(Lambda),num2str(Lambda2),
        num2str(Gap)};
2709
2710 % GNP
2711 [G]=FUNgnp(N,1/2);
2712 [GNPLambda,GNPEvector]=FUNpowermethod(G,200,0.001);
2713
2714 % calculate 2nd largest eigenvalue
2715 G1=G-GNPLambda*GNPEvector*GNPEvector';
2716 [GNPLambda2,GNPEvector2]=FUNpowermethod(G1,200,0.001);
2717
2718 GNPGap = GNPLambda - GNPLambda2;
2719 GNPD = {'GNP', '0',num2str(N),num2str(GNPLambda),num2str(GNPLambda2),
        num2str(GNPGap)};
2720 Data = [Data;GNPD];
2721
2722
2723 f = figure('Position',[10 10 500 700]);
2724 cnames = {'networkname','Sample#','Sample Size','Lambda_1','Lambda_2
        ','Gap',};
2725 t = uitable('Data',Data,'ColumnName',cnames,'parent',f,'Position',[1 1
         500 700]);
2726 filename = ['wiki_gap_vs_GNP'];
2727 saveas(gcf,filename);
2728
2729 close all;
2730
2731 %verify if it is a gen
2732 [LogA, SCodd] = FUNgen(Lambda, Evector);
```

```
2733 fileNM = ['wiki_gen'];
2734 saveas(gcf,fileNM);
2735
2736
2737 % calculate and plot wiki degree distribution and compare with GEOP
2738 function [] = wiki_vs_GEOP_DegDist(N, T, a, b, Seq, CompFlag,
         dimensionS, dimensionE,FilePath)
2739
2740 %-----------------output--------------------
2741 %-------------------------------------------
2742
2743 %-----------------input---------------------
2744 % N: Order of Graph
2745 % T: time T for GT
2746 % a,b: parameters of GEOP model
2747 % Seq: sequence num of samples with same parameters (N,a,b,dim,T)
2748 % CompFlag: if compare with GEOP model degdist
2749 %       1: yes
2750 %       0: no
2751 % dimensionS: dimension start num for loop
2752 % dimensionE: dimension end num for loop
2753 % FilePath: the degree distribution list file path,if it is null then
         take
2754 % the default value
2755 %-------------------------------------------
2756
2757 %////////////////////////////////////////////////////////////////////
2758
2759 % connect to specified database, specifying username and password
2760 connection = database ('OSN', 'root', 'mysql_root');
2761
2762 %found how many users are there in samples
2763
2764 Str1 = ['select degree, dist from wiki_degdist where seq=0 and flag =
         ''in''' ];
2765 cursor1 = exec(connection, Str1);
2766
2767 % set the reture data format as int/numeric
2768 setdbprefs('DataReturnFormat','numeric');
2769
2770 % retrieve indegree distribution
2771 cursor1 = fetch(cursor1);
2772 InDegree=cursor1.data(:,1);
2773 InDist=cursor1.data(:,2);
2774
2775 Str2 = ['select degree, dist from wiki_degdist where seq=0 and flag =
         ''out''' ];
2776 cursor2 = exec(connection, Str2);
2777
2778 % set the reture data format as int/numeric
2779 setdbprefs('DataReturnFormat','numeric');
2780
2781 % retrieve outdegree distribution
```

```
2782  cursor2 = fetch(cursor2);
2783  OutDegree=cursor2.data(:,1);
2784  OutDist=cursor2.data(:,2);
2785
2786  Str3 = ['select degree, dist from wiki_degdist where seq=0 and flag =
          ''all''' ];
2787  cursor3 = exec(connection, Str3);
2788
2789  % set the reture data format as int/numeric
2790  setdbprefs('DataReturnFormat','numeric');
2791
2792  % retrieve all degree distribution
2793  cursor3 = fetch(cursor3);
2794  AllDegree=cursor3.data(:,1);
2795  AllDist=cursor3.data(:,2);
2796
2797  hold on;
2798  %plot(log(InDegree), log(InDist), ':', log(OutDegree),log(OutDist),
          '-',log(AllDegree),log(AllDist), '-.');
2799  plot(log(InDegree), log(InDist), ':', log(OutDegree),log(OutDist),
          '-');
2800
2801  if CompFlag == 1
2802     [LStr,TStr]=GEOP_DegDist_plot(N, T, a, b, Seq, dimensionS,
             dimensionE,FilePath);
2803
2804      LegendStr = [substring(LStr,0,6),'''wiki indegree'',''wiki
             outdegree'',''wiki alldegree'',',substring(LStr,7,length(LStr)
             -1)];
2805      eval(LegendStr);
2806      titleStr = ['wiki  vs  ', TStr];
2807      fileNM = ['wiki_vs_GEOP_DegDist_' num2str(N) '_' num2str(T) '_'
             num2str(Seq) '_' num2str(a*100) '_' num2str(b*100) '_' num2str
             (dimensionS) '_' num2str(dimensionE) ];
2808  else
2809      legend('wiki indegree', 'wiki outdegree', 'wiki all degree');
2810      titleStr = ['wiki  log-log plot'];
2811      fileNM = ['wiki-degdist'  ];
2812
2813  end
2814
2815  title(titleStr);
2816  hold off;
2817  saveas(gcf,fileNM);
2818
2819  % calculate and plot wiki degree distribution and compare with GEOP
2820  function [] = wiki_vs_GEOP_DegDist(N, T, a, b, Seq, CompFlag,
          dimensionS, dimensionE,FilePath)
2821
2822  %-----------------output--------------------
2823  %-------------------------------------------
2824
2825  %-----------------input---------------------
```

```
% N: Order of Graph
% T: time T for GT
% a,b: parameters of GEOP model
% Seq: sequence num of samples with same parameters (N,a,b,dim,T)
% CompFlag: if compare with GEOP model degdist
%       1: yes
%       0: no
% dimensionS: dimension start num for loop
% dimensionE: dimension end num for loop
% FilePath: the degree distribution list file path,if it is null then
    take
% the default value
%-----------------------------------------

%////////////////////////////////////////////////////////////////////

% connect to specified database, specifying username and password
connection = database ('OSN', 'root', 'mysql_root');

%found how many users are there in samples

Str1 = ['select degree, dist from wiki_degdist where seq=0 and flag =
    ''in''' ];
cursor1 = exec(connection, Str1);

% set the reture data format as int/numeric
setdbprefs('DataReturnFormat','numeric');

% retrieve indegree distribution
cursor1 = fetch(cursor1);
InDegree=cursor1.data(:,1);
InDist=cursor1.data(:,2);

Str2 = ['select degree, dist from wiki_degdist where seq=0 and flag =
    ''out''' ];
cursor2 = exec(connection, Str2);

% set the reture data format as int/numeric
setdbprefs('DataReturnFormat','numeric');

% retrieve outdegree distribution
cursor2 = fetch(cursor2);
OutDegree=cursor2.data(:,1);
OutDist=cursor2.data(:,2);

Str3 = ['select degree, dist from wiki_degdist where seq=0 and flag =
    ''all''' ];
cursor3 = exec(connection, Str3);

% set the reture data format as int/numeric
setdbprefs('DataReturnFormat','numeric');

% retrieve all degree distribution
```

```
2875  cursor3 = fetch(cursor3);
2876  AllDegree=cursor3.data(:,1);
2877  AllDist=cursor3.data(:,2);
2878
2879  hold on;
2880  %plot(log(InDegree), log(InDist), ':', log(OutDegree),log(OutDist),
          '-',log(AllDegree),log(AllDist), '-.');
2881  plot(log(InDegree), log(InDist), ':', log(OutDegree),log(OutDist),
          '-');
2882
2883  if CompFlag == 1
2884     [LStr,TStr]=GEOP_DegDist_plot(N, T, a, b, Seq, dimensionS,
             dimensionE,FilePath);
2885
2886      LegendStr = [substring(LStr,0,6),'''wiki indegree'',''wiki
              outdegree'',''wiki alldegree'',',substring(LStr,7,length(LStr)
              -1)];
2887      eval(LegendStr);
2888      titleStr = ['wiki  vs  ', TStr];
2889      fileNM = ['wiki_vs_GEOP_DegDist_' num2str(N) '_' num2str(T) '_'
              num2str(Seq) '_' num2str(a*100) '_' num2str(b*100) '_' num2str
              (dimensionS) '_' num2str(dimensionE) ];
2890  else
2891      legend('wiki indegree', 'wiki outdegree', 'wiki all degree');
2892      titleStr = ['wiki  log-log plot'];
2893      fileNM = ['wiki-degdist'  ];
2894
2895  end
2896
2897  title(titleStr);
2898  hold off;
2899  saveas(gcf,fileNM);
2900
2901
2902
2903  % calculate wiki NL gap vs GEOP NL gap
2904  function [] = wiki_vs_geop_NL(N, T, a, b, Seq, dimensionS, dimensionE)
2905
2906  %-----------------output--------------------
2907  %-------------------------------------------
2908
2909  %-----------------input---------------------
2910  % N: Order of Graph
2911  % T: time T for GT
2912  % a,b: parameters of GEOP model
2913  % Seq: sequence num of samples with same parameters (N,a,b,dim,T)
2914  % dimensionS: dimension start num for loop
2915  % dimensionE: dimension end num for loop
2916  %-------------------------------------------
2917
2918
2919  Data = [];
2920  Datatxt = [];
```

```
Start = 1;
End = 1;
sampleStr = 'wikisample';

% initial GEOP parameters
% T = 0;
% a = 0.7;
% b = 0.15;
% Seq = 1;
% dimensionS = 1;
% dimensionE = 4;

geopN=N;

% check OS
OS = isunix;

sql = {};
for i = Start:End
    %generate wiki adjency matrix
    [K,A,u,outdegree,indegree,P] = FUNdbA('OSN', 'root', 'mysql_root',
         sampleStr,'0',sql);
    wikiN = length(A);

    [inL,outL, indegree, outdegree,inLambda_N, inLambda_2,outLambda_N,
         outLambda_2,inGap,outGap] = FUNLPgap(A);

    D = {sampleStr, num2str(wikiN),num2str(i),num2str(inLambda_N),
        num2str(inLambda_2),num2str(inGap),num2str(outLambda_N),
        num2str(outLambda_2),num2str(outGap)};
    Data = [Data;D];
    Datatxt = [Datatxt;0,wikiN,i,inLambda_N,inLambda_2,inGap,
        outLambda_N,outLambda_2,outGap];

    display('wiki is done');

    % calculate geop LP gap
    for j = dimensionS:dimensionE   %dimension
        GEOPsql1 = sprintf('select u,outdegree, indegree from
            geopsample where seq = %d and N= %d and dim = %d and a = %
            d and b = %d and T=%d order by userlabel', Seq,geopN,j,a,b
            ,T);
        GEOPsql2 = sprintf('select userlabel,friendlabel from
            geopsample_links where seq = %d and N= %d and dim = %d and
             a = %d and b = %d and T=%d order by userlabel', Seq,geopN
            ,j,a,b,T);
        GEOPsql ={GEOPsql1;GEOPsql2};
        % generate GEOP matrix according to N, a, b, dim, seq
        [K1,G] = FUNdbA('OSN', 'root', 'mysql_root', 'geopsample','1',
            GEOPsql);

        [inL,outL,indegree, outdegree,inLambda_N, inLambda_2,
            outLambda_N, outLambda_2,inGap,outGap] = FUNLPgap(G);
```

```
2961          GEOPD = {'GEOP', num2str(geopN),num2str(j),num2str(inLambda_N)
                  ,num2str(inLambda_2),num2str(inGap),num2str(outLambda_N),
                  num2str(outLambda_2),num2str(outGap)};
2962          Data = [Data;GEOPD];
2963          Datatxt = [Datatxt;1,geopN,j,inLambda_N,inLambda_2,inGap,
                  outLambda_N,outLambda_2,outGap];
2964
2965          str = sprintf('%dth GEOP is done',j);
2966          display(str);
2967
2968      end
2969
2970  end
2971
2972  filename = [sampleStr '_vs_GEOP_NL_gap_' num2str(geopN) '_' num2str(T)
          '_' num2str(Seq) '_' num2str(100*a) '_' num2str(100*b) '_'
         num2str(dimensionS) '_' num2str(dimensionE)];
2973  filenametxt = [filename '.txt'];
2974  [Flag] = FUNfileA(Datatxt,filenametxt);
2975
2976  if OS == 0    % windows (not unix)
2977      f = figure('Position',[10 10 700 700]);
2978      cnames = {'networkname','Sample Size','Sample#/dimension','
              inLambda_N','inLambda_2','inGap','outLambda_N','outLambda_2','
              outGap'};
2979      t = uitable('Data',Data,'ColumnName',cnames,'parent',f,'Position
              ',[1 1 500 700]);
2980      saveas(gcf,filename);
2981  end
2982
2983
2984  % calculate and plot youtube degree distribution
2985  function [] = youtube_DegDist(flag)
2986
2987  %-----------------output--------------------
2988  %-------------------------------------------
2989
2990  %-----------------input---------------------
2991  %frag: plot type
2992  %       0. 1 + 2
2993  %       1. degree distribution
2994  %       2. log-log
2995  %-------------------------------------------
2996
2997  % connect to specified database, specifying username and password
2998  connection = database ('OSN', 'amandatian', '1z3456');
2999
3000  % open cursor and issue SQL statement to select data
3001  cursor = exec(connection, 'select degree,dist from youtubedegree_dist
          order by degree');
3002
3003  % set the reture data format as int/numeric
3004  setdbprefs('DataReturnFormat','numeric');
```

```
3005
3006  % retrieve R rows of data
3007  cursor = fetch(cursor);
3008  Data=cursor.data;
3009  Degree = Data(:,1);
3010  Dist = Data(:,2);
3011
3012  % plot
3013
3014  if flag == 0
3015  subplot(1,2,1);
3016  plot(Degree, Dist);
3017  xlabel('k');
3018  ylabel('N(k)');
3019  title('Youtube-Degree Distribution');
3020
3021  subplot(1,2,2);
3022  plot(log(Degree), log(Dist));
3023  xlabel('log(k)');
3024  ylabel('log(N(k))');
3025  title('Youtube-Degree Distribution log-log plot');
3026  elseif flag == 1
3027      plot(Degree, Dist);
3028      xlabel('k');
3029      ylabel('N(k)');
3030      title('Youtube-Degree Distribution');
3031  elseif flag == 2
3032      plot(log(Degree), log(Dist));
3033      xlabel('log(k)');
3034      ylabel('log(N(k))');
3035      title('Youtube-Degree Distribution log-log plot');
3036  end
3037
3038
3039
3040  function [] = youtube_gap_gen(Start, End)
3041  % calculate youtube samples spectral gap, and verify if it follows gen
3042  %-----------------output--------------------
3043  %-------------------------------------------
3044
3045  %-----------------input---------------------
3046  % Start, End: loop of seq of samples
3047  %-------------------------------------------
3048  Data = [];
3049  Datatxt = [];
3050
3051  % check OS
3052  OS = isunix;
3053
3054  for i = Start:End
3055      %generate adjency matrix
3056      [K,A] = FUNdbA('OSN', 'root', 'mysql_root', 'youtubesample',
          num2str(i),[]);
```

```
3057      N = length(A);
3058      % calculate dominent eigenvector by power method
3059      [Lambda,Evector, count]=FUNpowermethod(A,50,0.001);
3060
3061      % calculate 2nd largest eigenvalue
3062      A1=A-Lambda*Evector*Evector';
3063      [Lambda2,Evector2,count2]=FUNpowermethod(A1,50,0.001);
3064
3065      %[Lambda,Lambda2,Evector] = eigen(A);
3066
3067      Gap = Lambda - Lambda2;
3068      D = {'youtubesample', num2str(i),num2str(N),num2str(Lambda),
              num2str(Lambda2),num2str(Gap)};
3069      Data = [Data;D];
3070      Datatxt = [Datatxt;0,i,N,Lambda,Lambda2,Gap];
3071
3072      % GNP
3073      [G]=FUNgnp(N,1/2);
3074      [GNPLambda,GNPEvector]=FUNpowermethod(G,200,0.001);
3075
3076      % calculate 2nd largest eigenvalue
3077      G1=G-GNPLambda*GNPEvector*GNPEvector';
3078      [GNPLambda2,GNPEvector2]=FUNpowermethod(G1,200,0.001);
3079
3080      GNPGap = GNPLambda - GNPLambda2;
3081      GNPD = {'GNP', num2str(i),num2str(N),num2str(GNPLambda),num2str(
              GNPLambda2),num2str(GNPGap)};
3082      Data = [Data;GNPD];
3083      Datatxt = [Datatxt;1,i,N,GNPLambda,GNPLambda2,GNPGap];
3084
3085
3086      if OS == 0 % windows/ not unix operation system
3087          %verify if it is a gen
3088          [LogA, SCodd] = FUNgen(Lambda, Evector);
3089          fileNM = ['youtube_gen_' num2str(i)];
3090          saveas(gcf,fileNM);
3091      end
3092  end
3093
3094  filename = ['youtube_gap_' num2str(Start) '_' num2str(End)];
3095  filenametxt = [filename '.txt'];
3096  [Flag] = FUNfileA(Datatxt,filenametxt,'dec');
3097
3098  if OS == 0    % windows (not unix)
3099      f = figure('Position',[10 10 500 700]);
3100      cnames = {'networkname','Sample#','Sample Size','Lambda_1','
              Lambda_2','Gap',};
3101      t = uitable('Data',Data,'ColumnName',cnames,'parent',f,'Position
              ',[1 1 500 700]);
3102      saveas(gcf,filename);
3103  end
3104
3105
```

```
% choose a youtubesampe and calculate NL gap compare with GEOP NL gap
function [] = youtube_vs_geop_NL(T, a, b, dimensionS, dimensionE,
    sampleSeqS,sampleSeqE,flag)

%-----------------output--------------------
%-------------------------------------------

%-----------------input---------------------
% T: time T for GT
% a,b: parameters of GEOP model
% dimensionS: dimension start num for loop
% dimensionE: dimension end num for loop
% sampleSeqS,sampleSeqE: choose youtube sample seq(for loop)
% flag: type of algorithm about T
%       1: GEOP
%       2: Age
%       3: degree
%-------------------------------------------
Data = [];
Datatxt = [];
sampleStr = 'youtubesample';

% T = 1;
% a = 0.7;
% b = 0.15;
% dimensionS = 1;
% dimensionE = 4;

% check OS
OS = isunix;
sql = {};
for i = sampleSeqS:sampleSeqE
    %generate adjency matrix
    [K,A,u,outdegree,indegree,P] = FUNdbA('OSN', 'root', 'mysql_root',
         sampleStr,num2str(i),sql);
    N = length(A);

    [inL,outL, indegree, outdegree,inLambda_N, inLambda_2,outLambda_N,
         outLambda_2,inGap,outGap] = FUNLPgap(A);

    D = {sampleStr, num2str(N),num2str(i),num2str(inLambda_N),num2str(
        inLambda_2),num2str(inGap),num2str(outLambda_N),num2str(
        outLambda_2),num2str(outGap)};
    Data = [Data;D];
    Datatxt = [Datatxt;0,N,i,inLambda_N,inLambda_2,inGap,outLambda_N,
        outLambda_2,outGap];

    for j = dimensionS:dimensionE   %dimension
        G=[];
        GT=[];

        %[G,GT,degree] = FUNgeoP(a,b,N,j,T);
        if flag ==1
```

```
3153              [G,GT] = FUNgeoP(a,b,N,j,T);
3154          elseif flag ==2
3155              [G,GT] = FUNgeoPAge(a,b,N,j,T);
3156          elseif flag ==3
3157              display('not defined algorithm for T');
3158          else
3159              display('not defined algorithm for T');
3160          end
3161
3162          [inL,outL,indegree, outdegree,inLambda_N, inLambda_2,
                  outLambda_N, outLambda_2,inGap,outGap] = FUNLPgap(GT);
3163          GEOPD = {'GEOP', num2str(N),num2str(j),num2str(inLambda_N),
                  num2str(inLambda_2),num2str(inGap),num2str(outLambda_N),
                  num2str(outLambda_2),num2str(outGap)};
3164          Data = [Data;GEOPD];
3165          Datatxt = [Datatxt;1,N,j,inLambda_N,inLambda_2,inGap,
                  outLambda_N,outLambda_2,outGap];
3166      end
3167
3168  end
3169
3170  filename = [sampleStr '_vs_GEOP_NL_gap_' num2str(N) '_' num2str(T) '_'
          num2str(100*a) '_' num2str(100*b) '_' num2str(dimensionS) '-'
         num2str(dimensionE)];
3171  filenametxt = [filename '.txt'];
3172  [Flag] = FUNfileA(Datatxt,filenametxt);
3173
3174  if OS == 0    % windows (not unix)
3175      f = figure('Position',[10 10 700 700]);
3176      cnames = {'networkname','Sample Size','Sample#/dimension','
              inLambda_N','inLambda_2','inGap','outLambda_N','outLambda_2','
              outGap'};
3177      t = uitable('Data',Data,'ColumnName',cnames,'parent',f,'Position
              ',[1 1 500 700]);
3178      saveas(gcf,filename);
3179  end
3180
3181  -- ------------------------------------------------------
3182  -- code to call SNAP methods to calculate diameter
3183  -- ------------------------------------------------------
3184  #include "stdafx.h"
3185
3186  int main(int argc, char* argv[] ) {
3187
3188          //get parameters
3189          Env = TEnv(argc, argv, TNotify::StdNotify);
3190          const TStr InFNm = Env.GetIfArgPrefixStr("-i:", "graph.txt", "
                  Input file");
3191          const TStr OutFNm = Env.GetIfArgPrefixStr("-o:", "output.txt",
                   "Output file");
3192          const int MaxNode = Env.GetIfArgPrefixInt("-m:", 1000000, "
                  Maximum Node Count");
3193
```

```
3194          //generate graph by input txt file with edge links
3195          PNGraph Graph = TSnap::LoadEdgeList<PNGraph>(InFNm, 0, 1);
3196          int A;
3197          int B;
3198
3199          B=Graph->GetNodes();
3200
3201          //get diameter of G
3202          A = TSnap::GetBfsFullDiam(Graph, MaxNode, true);
3203
3204          //write diameter to file
3205          char s[] = "the diameter of graph is";
3206          char c = '\n';
3207          FILE *stream;
3208
3209          stream = fopen( OutFNm.GetCStr(), "w" );
3210          fprintf( stream, "%s%c", s, c );
3211          fprintf( stream, "%d\n", A );
3212          fprintf( stream, "%d\n", B );
3213          fclose( stream );
3214  }
3215
3216  -- ------------------------------------------------------
3217  -- code to call SNAP methods to calculate eigenvalues
3218  -- ------------------------------------------------------
3219  #include "stdafx.h"
3220
3221  int main(int argc, char* argv[]) {
3222
3223
3224          //get parameters
3225          Env = TEnv(argc, argv, TNotify::StdNotify);
3226          const TStr OutFNm = Env.GetIfArgPrefixStr("-o:", "output.txt",
                   "Output file");
3227          const int Order = Env.GetIfArgPrefixInt("-N:", 100, "Graph
                  Order");
3228          const int Edge = Env.GetIfArgPrefixInt("-E:", 1000, "Graph
                  Edge");
3229
3230      // create a graph
3231          PNGraph Graph = TSnap::GenRndGnm<PNGraph>(Order, Edge);
3232          // convert to undirected graph TUNGraph
3233      PUNGraph UGraph = TSnap::ConvertGraph<PUNGraph>(Graph);
3234
3235
3236
3237          int Norder;
3238          Norder=UGraph->GetNodes();
3239
3240          //calculate eigenvalue, fist two largest and first two
                  smallesa
3241          TFltV EigValV;
3242          TVec<TFltV> EigV;
```

```
3243          TSnap::GetEigVec(UGraph, 4, EigValV, EigV);
3244
3245          //write data into file
3246          FILE *stream;
3247
3248          stream = fopen( OutFNm.GetCStr(), "w" );
3249          fprintf(stream,"Eigen Values:\n");
3250
3251          for (int i =0; i < EigValV.Len(); i++)
3252          {
3253                  fprintf(stream, "%d\t%f\n", i, EigValV[i]());
3254          }
3255
3256          fclose( stream );
3257  }
```

Bibliography

[1] L.A. Adamic, O. Buyukkokten, E. Adar, A social network caught in the Web, *First Monday* **8** (2003) 440-442.

[2] Y. Ahn, S. Han, H. Jeong, H. Kwak, S. Moon, Analysis of topological characteristics of huge on-line social networking services, In: *Proceedings of the 16th International Conference on World Wide Web*, 2007.

[3] A. Bonato, *A Course on the Web Graph*, American Mathematical Society Graduate Studies Series in Mathematics, Providence, Rhode Island, 2008.

[4] R. Babu, *Numerical Methods*, Dorling Kindersley (India) Pvt. Ltd., 2010.

[5] A. Barabási, R. Albert, Emergence of scaling in random networks, *Science* **286** (1999) 509-512.

[6] B. Bhattacharjee, P. Druschel, M. Marcon, A. Mislove, Measurement and analysis of on-line social networks, In: *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, 2007.

[7] A. Bonato, J. Janssen, P. Prałat, A geometric model for on-line social networks, In: *Proceedings of 3rd Workshop on Online Social Networks*, 2010.

[8] F.R.K. Chung, Spectral Graph Theory, *American Mathematical Society* 1997.

[9] F.R.K. Chung, L. Lu, Complex Graphs and Networks, *American Mathematical Society* 2004.

[10] E. Estrada, Spectral scaling and good expansion properties in complex networks, *Europhys. Lett.* **73** (2006) 649-655.

[11] B. Huberman, S. Golder, D. Wilkinson, Rhythms of social interaction: messaging within a massive on-line network, In: *3rd International Conference on Communities and Technologies*, 2007.

[12] R. Kumar, J. Novak, A. Tomkins, Structure and evolution of online social networks, In: *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2006.

[13] D. Liben-Nowel, An algorithmic approach to social networks, Ph.D. Dissertation, Massachusetts Institute of Technology, 2005.

[14] List of social networking websites, Wikipedia. Accessed July 1, 2011. `http://en.wikipedia.org/wiki/List_of_social_networking_websites`.

[15] S. Milgram, The small world problem, *Psychology Today* **2** (1967) 60-67.

[16] Online Social Network Research@The Max Planck Institute for Software Systems. Accessed May 10, 2010. `http://socialnetworks.mpi-sws.org/data-imc2007.html`.

[17] Press Room, Facebook. Accessed July 1, 2011. `http://www.facebook.com/press/info.php?statistics`.

[18] Stanford large network dataset collection. Accessed July 5, 2010. `http://snap.stanford.edu/data/index.html`.

[19] The History of Usenet, Usenet.com. Accessed July 1, 2011. `http://www.usenet.com/usenet_history.html`.

[20] Usenet, Wikipedia. Accessed July 1, 2011. `http://en.wikipedia.org/wiki/Usenet`.

[21] D. J. Watts, S. H. Strogatz, Collective dynamics of "small-world" networks, *Nature* **393** (1998) 440-442.