Concrete Graph Mining: a Database approach

Wei Chen
Dept. of Computer Science
CMU
weichen1@andrew.cmu.edu

Siping Ji
Dept. of Computer Science
CMU
sipingji@cmu.edu

November 30, 2013

Abstract

In this study, we focus on how to implement various graph mining algorithms based on RDBMS. Since the introduction of MapReduce, it has been considered the only silver bullet in large scale data analysis. While from database's perspective, RDBMS has been equipped with all the requierd capabilities for over twenty years. So why not RDBMS? We want to justify RDBMS's role in future big data application by experiment with current graph mining algorithms using RDBMS. The main contribution of this study is to explore the limitation in RDBMS's expressiveness and scalability, which scenario RDBMS will be a more appropriate alternative than MapReduce.

1 Introduction

The problem we want to solve is the following:

- GIVEN: a graph stored as edge list in RDBMS
- FIND: hidden patterns in this graph using various graph mining algorithms implemented by SQL language
- to MINIMIZE: the computational cost.

With the growth of computer network, graphs are ubiquitous nowadays, social networks, papers citing network, world wide web to a few. On the other hand, with the drastic drop of storage cost and the emergence of large social network companies like Facebook and Linkedin, graphs are now of a unprecedented size with billions of vertexes and edges. Mining such large dataset may help us gain lots of useful information and leads to interesting applications anomaly detection, social network analysis and so on. There are also lots of famous graph mining algorithms like PageRank, Random Walk with Restart, Belief Propagation who are aiming to find interesting patterns in graphs. However, most of these algorithms assume the graph fit in memory, which is apparently no longer suitable for the giant graphs today.

In this paper, we focus on how to use SQL language to implement efficient graph mining algorithms to find interesting patterns in large graphs that are stored in relational databases. We want to justify that RDBMS, equipped with highly optimized query engine and in-memory index, is sufficient for us to develop efficient and scalable graph mining algorithms in order to deal with graphs in this big data era.

2 Survey

Next we list the papers that each member read, along with their summary and critique.

2.1 Papers read by Siping Ji

The first paper was the "PEGASUS" paper by Kang [6]

- *Main idea*: This paper proposes a parallel programming primitive GIM-V that captures the shared characteristics among many seemingly different graph mining algorithms. The key idea is that the computation of graph mining algorithms like Pagerank, RWR, diameter estimation and so on, are essentially iterations of a generalized matrix-vector multiplication. More specifically, this generalized matrix-vector computation step can be captured by three basic operators in GIM-V - combine2, combineAll and assign. Based on this abstraction, many graph mining algorithms can be effciently computed via GIM-V over hadoop. Further, the author addresses the problem of high computational overhead of the shuffling stage in map-reduce. To overcome the problem and thus enhance the scalability, the author introduces several techniques, namely blocking, edge-clustering, diagonal block iteration and node renumbering into the implementation of GIM-V. Then the author conducts experiments showing that these optimization techniques siginificantly improves the scalability of the proposed method.
- *Use for our project*: Although GIM-V is a hadoop-based implementation, it is still a good reference for us since matrix-vector multiplication is easy to implement under SQL. We can also implement the operator primitives of GIM-V(combine2, combineAll and assign) using user-defined function in PostgreSQL.
- *Shortcomings*: This paper only compares the performance of GIM-V with different optimization techniques, but lack the comparison with other graph mining models or methods.

The second paper was "Patterns on the Connected Components of Terabyte-Scale Graphs" by Kang [4]

- *Main idea*: This paper studies and analyzes the patterns in connected components in real world large graphs. By introducing the concept of Graph Fractal Dimension(GFD) as a measure of the density of a connected component and the maximum effective radius(MER) vs. average effective radius(AER) ratio, the author shows there exists both consistency and difference among connected components of different size. Besides, the author also suggests an log-linear relationship between rebel probability of a newcomer node and its degree in dynamically evolving graphs. Based on these observations, the author proposes a Community Connection model to explain the growth process of a graph and justifies that this model correctly captures the patterns they discovered.

- *Use for our project*: It is very relevant to the task of implementing the connected component algorithm. Additionally, this paper exemplifies how to discover, analyze and model the hidden pattern in a graph instead of treating statistics only as useless numbers. I think this helps us better explain the result in our experiment process.
- *Shortcomings*: Though this paper discusses extensively about analyzing and model patterns of connected components of a graph, it doesnt address much about how to apply this pattern and model to applications. Also, the implementation detail of finding connected components is ignored in the paper, which is what our project mainly focuses.

The third paper was "Inference of Beliefs on Billion-Scale Graphs" by Kang [3]

- *Main idea*: Belief Propagation(BP) is an popular graphical model algorithm for inferring the states of nodes in Markov Random Fields, it has been successfully applied to many problems in the fields of social network analysis, computer vision and so on. There already exists many efficient BP algorithms, but all of them assumes the graph can fit in main memory. This paper addresses exactly the problem of how to scale up the belief propagation(BP) algorithm to giant graphs with billions of nodes that can only stored on disks. The author first proposes the GIM-V - a primitive for parallel graph mining algorithms based on hadoop. It is based on the observation that the computation process of many existing graph mining algorithm like pagerank, random walk with restart are essentially repeated matrix vector multiplication by customizing the sub-operations in a matrix multiplication. Next, the author shows that by converting the original graph to a directed line graph, the key step in the BP computation that updates the messages vectors can be viewed as a process of matrix-vector multiplication, and thus can be applied to the framework of GIM-V. To further improve the efficiency of the parallel computing algorithm, the author introduces a trick of lazy multiplication to reduce the computation cost of multiplication operations. In the experiment, it shows that the method proposed beat the single machine BP algorithm in terms of running time, and when the number of machines increases, the algorithm scales up near linearly.
- *Use for our project*: Although the algorithm proposed in this paper is based on hadoop, it's still insightful to see the reinterpretation of the process of computing BP as a matrix-vector multiplication step. This insight may inspire us to efficiently compute BP in RDBMS, since the computation of sparse matrix-vector multiplication can be very efficient due to highly optimized query execution and in-memory index.
- *Shortcomings*: The major weakness of this paper is that it only compare their method to single-machine BP algorithm. It also lacks the theoretic justification for the near-liner scalability of their algorithm.

2.2 Papers read by Wei Chen

The first paper was "Mining Large Graphs: Algorithms, Inference, and Discoveries". [2]

- *Main idea*: This paper discuss how to do inference in graphical model under distributed setting, what if graph can not fit into main memory. They propose a variant of Belief Propagation called *Line Graph Fixed Point(LFP)* to address this issue. The key step is to induce a new graph called *Directed Line Graph* L(G) from original graph, it flips the definition of node and edge. Each node of L(G) represents an edge of G and edge exists between node n_i and n_j if and only if their related edges are incident. L(G) has a nice property that it's easy to derive exact recursive equation for it, actually the message updating equation of original Belief Propagation [15] can be adopted without modification. Then the author generalize mesage updating procedure as general Matrix-Vector operations, namely combine2(), combineAll(), sumVector() and assign() from [6]. While directly apply LFP without careful design will lead to drastic storage requirement to store L(G), the idea to tackle this is *Lazy Multiplication*, all operations on L(G) is is actually run on G. They propose an algorithm employing this idea, and run on Hadoop, its name is *Hadoop Line Graph Fixed Point*. Ha-LFP solves the biggest issue of the most graph mining algorithm — scalability. Since Ha-LFP is based on Hadoop, it inherits fault tolerance, data replication natively.
- *Use for our project*: LFP seems like a promising method to do Belief Propagation, combine2, combineAll, assign is natural in SQL. Matrix operation is tractable in SQL, and can be optimized by RBDMS. The performance could be comparable to Hadoop version. Also we don't need to construct L(G) explicitly, this avoids expensive cost in storage and time to access and update L(G).
- *Shortcomings*: It's kind of tricky to say this algorithm solve scalability perfectly. I think this is more contributed to Hadoop instead of LFP. There may exist better formulation of Belief Propagation on Hadoop. The experiments are conducted on M45, which is one of the most advanced supercomputer.

The second paper was "Understanding Belief Propagation and its Generalizations". [15]

- *Main idea*: Marginal Probability is important in graphical model inferencing, while most of the time it is expensive to calculate, the running time is usually exponential to the number of nodes. *Belief Propagation* is a neat technique to bring the running time down to linear to the number of edges in graphical model, and it has been widely used beyong machine learning, like computer vision[1], turbo code [11], etc. The success of Belief Propagation results from a property called *Conidtional Independence*. Most graphical models have the property that there exist many conditional independence between nodes. The core idea of Belief Propagation is to exploit this property to decouple the computation of global probability into several local computations. The marginal probability of node n_i is called *belief*. Each node n_i sends messages to other nodes n_j. *Message* $m_{ij}(n_j)$ represents how node n_i thinks node n_j's state should be. A node can calculate its belief once it has received messages from all its neighbouts. So each edge is associate with two messages. The paper shows that each message can be calculated only once, thus we can use dynamic programming to compute every node's belief in linear time. The paper also introduce Generalized Belief Propagation which extends BP to group of nodes.

- *Use for our project*: This paper discusses how to compute belief of each node in one round(for trees). We can first implement the algorithms mentioned in this paper as a baseline. It is also helpful for understanding other optimizing algorithms for Belief Propagation.
- *Shortcomings*: There is not much discussion about the implementation in SQL. But user defined functions provided by PostgreSQL has limited expressiveness, so the real implementation will be quite different.

The third paper was "A Comparison of Approaches to Large-Scale Data Analysis". [13]

- *Main idea*: This paper compares the tradeoff between Parallel DBMS and MapReduce framework in large data analysis. The authors want to advocate the capability of Parallel DBMS in implementing large scale analytic tasks. The paper analyze the fundamental difference of Parallel DBMS and MapReduce, their support in schema, data distribution, fault tolerance, index, and utility tools. Then they conduct experiments about their performance in different tasks, like pattern searching, aggregation, join, etc. They conclude that Parallel DBMS outperforms MapReduce in all kinds of tasks. But Parallel DBMS requires a lot of effort to configure and profile to reach good performance, MapReduce is relatively much easier to use. In the end, the paper admit MapReduce's advantage over complex Parallel DBMS, while still suggest Parallel DBMS as an option for specific tasks.
- *Use for our project*: We can borrow some ideas from theirs experiments about investigate SQL function. Their analysis method is also a good model to follow.
- *Shortcomings*: The tasks they experimented are kind of trivial. We don't know the real difference in when the systems are used for more complex graph mining algorithms.

The fourth paper was "Pig Latin: A Not-So-Foreign Language for Data Processing". [12]

- *Main idea*: Either RDBMS or MapReduce framework represents some extreme in large scale data analysis, either unnatural for programmers' mind(SQL) or too low level to express many algorithms(MapReduce). Different solutions are proposed. Pig is a platform of Yahoo that try to reach a balance between these two styles to support more flexible(ad-hoc) data analysis tasks. It takes a hybrid strategy, provides friendly user interface(programming, debugging), while actual tasks are running on powerful Hadoop platform without loss of performance. And it propose a new high level programming language called *Pig Latin* that gives user more control of data flow, which programmer can adopt an imperative style, while retain the expressiveness of SQL. It also support several key operations that can be automatically parallelized, like filter, cogroup, group, join, etc. Its underlying implementation is compile Pig Latin programs into Hadoop jobs, so that it is also equipped with the benefits of MapReduce.

- *Use for our project*: Pig Latin's SQL-like syntax provides finer grained control over dataflow, which is more suitable for implementing graph mining algorithms. We can compare with SQL user defined function implementation of these algorithms, and analyze their intrinsic data accessing characteristic.
- *Shortcomings*: There are few graph mining algorithms implementation in Pig Latin, only a few open source analysis package like Linkedin's Datafu. And Pig only supports read-only data analysis workloads, which may lead to many unnecessary efforts if we want to do extensive write during computation.

The fifth paper was "Fast counting of triangles in large real networks without counting: Algorithms and laws" [14].

- *Main idea*: The paper proposed a analytical method to count the number of triangles in social networks. It is based upon two theorems regarding the fundamental property of adjencency matrix. The first theorem is that the number of global triangles is proportional to the sum of cubes of its eigenvalues, which is more formally defined as follow:
$$\Delta(G) = \frac{1}{6}\sum_{i=1}^{n} \lambda_i^3 \tag{1}$$
Another one is about local triangle, the formula for its count is as follows:
$$\Delta_i = \frac{\sum_j u_{ij}^2 \lambda_j^3}{2} \tag{2}$$
The author conducted several experiments on different datasets. and discovered several patterns using visualization, such as power law.
- *Use for our project*: For task 7, we adopt the algorithms described in the paper to count the number of global triangles and local triangles. The algorithm to calculate eigenvalues and eigenvectors is Lanczos, which is well suited for solving approximate top-k eigenvalues and eigenvectors.
- *Shortcomings*: The paper didn't discuss the selection of k, which is the number of top eigenvalues use for calculation. So most implementations have to randomly pick a small number and verify its effect. And how the magnitude of k will affect the running time of the algorithm.

3 Proposed Method

We implement all the graph mining algorithms using Postgres's embedded PL/pgSQL programming language, which supports many advanced features, like user defined function, aggregate, etc. It also has a sophisticated query execution engine, which we think is the most critical component of Postgres.

The following are the algorithms we plan to implement. The reason we choose them is that there are a lot of implementation in other platform like MapReduce, we can compare our SQL version with them to draw a clusion about SQL's unique charastic in solving data analytic tasks.

Degree Distribution: Plot the distribution of each node's degree.

PageRank: Determine the importance of every node of a graph based on its connectivity.

Connected Components: Partition the nodes of a graph also based on its connectivity.

Radius of Every Node: Compute the radius of every node in a graph. The radius is defined as the number of hops that a node needs to reach to its furthest neighbor.

Belief Propagation: Calculate the marginal probability of each node in a graphical model.

Eigenvalue: Using approximation method to estimate the top-k eigenvalues of a matrix.

Count of triangle Using approcimate method to calculate the number of triangles in a social network.

Shortest Path Calculate the shortest path from each node in a graph to a single source node.

Minimum spanning tree Construct a tree which is a subgraph of original graph with the minimum sum of weight of its edges.

3.1 Degree Distribution

We employ Postgres's group by command according to either source node or target node to count the degree distribution according to each node. The pseudo code is in Algorithm 1, 2.

Algorithm 1 Out Degree distribution

```
Group edges according to source node's id
Count the number of members of each group
```

Algorithm 2 In Degree distribution

```
Group edges according to target node's id
Count the number of members of each group
```

3.1.1 Math

First we need to count degree of each node. Then we count the frequency of each degree count.

3.1.2 Idea of SQL implementation

The only operation we need from SQL is its group by clause. We can aggregate the edges according with source node or target node.

3.1.3 SQL code

Please refer to the code in **Appendix**.

3.2 Pagerank

The algorithm of pagerank is Power method. We do matrix multiplication continuously until the change in pagerank is small. The most important equation for calculating pagerank is as follows:

$$PR(i) = \alpha\frac{1}{N} + (1-\alpha)\sum_{j \in InNeighbor(i)} \frac{PR(j)}{OutDegree(j)} \tag{3}$$

Algorithm 3 Pagerank

Bulk load graph into an edge table in database.
Initialization(damper factor=0.85, max iteration = 100, epsilon = 0.0001)
Build a weight matrix trans, initialize pagerank p
repeat
  For each node i, update its pagerank with its old value and its income node's pagerank.
until Convergence

3.2.1 Math

According to the definition of pagerank, we can gain an intuitive idea of how to calculate the pagerank of a node. We just take weighted sum of its incoming neighbors. We can encode this operation as matrix vector multiplication. We can do this multiple times. And the fix point of the equation is the pagerank of the graph. And by the results from linear algebra, we know that the stable vector is the eigenvector with biggest eigenvalue. Thus we can get the eigenvector by power method, which just do multiplication until convergence.

3.2.2 Idea of SQL implementation

The implementation consists of several components. First, we have a *graph* which has the schema *(from_id, to_id, weight)*. Then we will build a *rank* table has the schema *(node_id, rank)*, all the rank are initialized randomly. After we enter loop, we will do a large matrix vector multiplication, which is implemented by SQL *select* and *join*. The new pagerank is stored in a temporary table. After the loop is over, we calclate the updates to every node, if the change is little, then abort.

3.2.3 SQL code

Please refer to the code in **Appendix**.

3.3 Weakly connected components

In terms of the implementation of weakly connected components. We borrow the idea of HCC method from the "PEGASUS" paper.[6]

3.3.1 method

The key idea of this algorithm is that for every node v_i in the graph, we maintain a component id c_i^h which is the minimum node id within h hops from v_i. Initially, c_i^h of v_i is set to its own node id. For each iteration, each node sends its current component id to its neighbors. Then c_i^{h+1} is set to the minimum value among its current component id and the received component ids from its neighbors. Finally, when the update converges, all nodes in the same connected component will share the same component id.

3.3.2 Idea of SQL implementation

The algorithm can be described in algorithm 4. The key step that updates the component id to the minimum of its neighbors' is accomplished in SQL using join and group by clause. After several rounds of iteration, the nodes in the same connected component will share the

Algorithm 4 Weakly Connected Component

```
Bulk load graph into an edge table in database.
Create a component table, where each entry contains a node id, and the component id.
Initialize the component table where component id equals node id.
repeat
    For each node, assign the minimum component id of its neighbors as the new component
    id of this node.
until Convergence
```

same component id. The number of iterations for convergence can be proved to be upper bounded by the diameter of the graph.

3.3.3 SQL code

Please refer to the code in **Appendix**.

3.4 Radius of every node

We discard the traditional algorithm because it is extremely infeasible for large graphs, since it uses a set to record every neghbors within n hops for a node during iteration, which requires a O(n^2) space.

3.4.1 method

Since the exact algorithm is hopeless, we use the approximation algorithm described in "HADI" paper[7] instead. Specifically, we use the Flajolet-Martin algorithm for counting the number of distinct members in a multiset. It is guaranteed to give an unbiased estimate and a tight O($log(N)$) bound for space complexity. The basic idea of Floajolet-Martin algorthm is to use a bitstrings of length L to encode the set. For each element to add, we randomly pick up a index according to a specified distribution, and assign BITMAP[index] to 1. Following this procedure to add element, the size of the final set can be estimated by $\frac{1}{\phi}2^{\frac{1}{k}\sum_{i=1}^{k} R_i}$, where $\phi = 0.77351$, R_i denotes the index of the leftmost 0 in the the kth bitstring.

In our proposed method for computing radius of every node, we use the Flajolet-Martin(FM) bitstrings to encode the neighbors of every node. Formally, we use k FM-bitstrings $b(h, i)$ to represent the set of neighbor nodes reachable from $node_i$ within h hops. And for each iteration, we use the following way to update each FM-bitstring:

$$b(h,i) = b(h-1,i) \quad BIT-OR \quad b(h-1,j)|(i,j) \in E$$

Given the above description of how to encode neighbors of a node, and the method to update bitstrings, we can describe the approximation method we used to compute the raidus of every node in algorithm 5.

Algorithm 5 Radius of Every Node

Bulk load graph into an edge table in database.
Preprocess the edge table, add a self loop edge to every node in the graph.
Initialize the vertex table, which contains node id, and a column of bitstring array, the bitstrings are initialized using the FM algorithm
repeat
    For every node update the bitstring according to formula: $b(h,i) = b(h-1,i) \quad BIT-OR \quad b(h-1,j)|(i,j) \in E$.
    For every node, check whether the bitstrings is unchanged before and after updates, if it's not changed, output i as the radius for this node.
until The bitstrings of every node stabilizes or it reaches the maximum rounds of iteration.

3.4.2 Idea of SQL implementation

In order to store the fm-string array in the table, we use the array type which is supported by PostgreSQL. To initialize the fm string and update the fm string array, we defined some user defined functions in PostgreSQL. The key step of algorithm that updates the FM-bitstring array is accomplished by using aggBitOr in the join and group by clause. Specifically, we join the edge table and vertex table on dst id, group by src id, and then call the aggBitOr to update the fm string array for all nodes. We summarize the user defined functions in Table 1.

Table 1: User defined functions for task 4

function name	description
fmAssign	Assign the k FM-bitstrings for a node
bit-or	Execute the OR operations between two bitstring arrays
aggBitOr	Aggregate function for bit-or
fmSize	Estimate the size of a set encoded by FM-bitstring

3.4.3 SQL code

Please refer to the code in **Appendix**.

3.5 Eigenvalue

We adopt the method propose in [5]. There are several methods to solve part of eigenvalue computation problems, for instance, power method[10]. While it has the limitation that it can only extract the eigenvector with biggest eigenvalue. Several method have been proposed to extract top k eigenvectors simultaneously. The approach we use is Lanczos algorithm[9]. The general idea about this algorithm is that instead of directly work on an $N \times N$ matrix, we first generate a skinny $N \times m$ matrix(M ≪ N). Then it computes a small $M \times M$ dense matrix which has good approximation to the eigenvalues of the original matrix. In this case, we directly apply quadratic algorithm to top-k eigenvalues. Notice that k < M.

Algorithm 6 Lanczos algorithm

Input: Matrix $A^{n \times m}$
random n-vector b,
number of steps m
output: Orthogonal matrix $V_m^{v \times m} = [v_1 \cdots vm]$,
coefficients $\alpha[1..m]$ and $\beta[1..m-1]$

```
1: β_0 ← 0, v_0 ← 0, v_1 ← b/||b||
2: for i = 1 to m do
3:     v ← A v_i
4:     α_i ← v_i^T v
5:     v ← v − β_{i−1} v_{i−1} − α_i v_i
6:     β_i ← || v ||
7:     if β_i = 0 then
8:         break for loop
9:     end if
10:    v_{v+1} ← v/β_i
11: end for
```

Algorithm 7 Build tridiagonal matrix

Input: α, β **Output:** $T_m^{m \times m}$

```
for i = 1 to m do
    T[i,i] ← α_i
    T[i,i+1] = T[i+1,i] ← β_i
end for
```

Algorithm 8 Compute Ritz values

Input: Orthogonal matrix $V_m^{n \times m}$
coefficients $\alpha[1..m]$ and $\beta[1..m-1]$

```
T_m ← (build a tridiagonal matrix from α and β)
QDQ^T ← EIG(T_m)
λ_{1..k} ← (top k eigenvalues from D)
Q_k ← (k columns of Q corresponding to λ_{1..k})
R_k ← V_m Q_k
```

3.5.1 Math

Different from power method, the intermediate multiplication matrix is used to construct a set of orthonormal base of *Krylv subspace* K_m which follows the definition:

$$K_m =< b, Ab, \cdots, A^{m-1}b > . \tag{4}$$

The sub procedure to construct orthonormal bases may be any standard algorithm, for example Gram-schmidt algorithm. We can view Lanczos algorithm as an iterative method which incrementally construct Krylov subspace. The pseudo-code is shown in Algorithm 3.5.

After Lanczos factorization, we get a few matrices that satisfy the following equation:

$$AV_m = V_m T_m + f_m e_m^T \tag{5}$$

To name a few, $A^{n \times m}$ is input matrix, $V_m^{n \times m}$ contains the m orthonormal bases, $T_m^{m \times m}$ is a tridiagonal matrix, f_m is new n-vector orthogonal to all columns of V_m, e_m is a vector that mth element is 1, and others 0. After algorithm 3.5, we need to construct the matrix $T_m^{m \times m}$. The algorithm is quite simple, it is listed in algorithm 7.

The eigenvalues of T_m are called Ritz values, and $V_m Y$'s columns are called Ritz vector. It is constructed by Algorithm 8. We expect the Ritz values and Ritz vectors to be good approximation of the eigenvalues and eigenvectors of original matrix. The computation of eigenvalues of T_m can be done by standard quadratic algorithms, such as QR method.

3.5.2 Idea of SQL implementation

Since the algorithm of Lanczos algorithm is matrix calculation intensive, so we wrap all matrix related operation in our host language Python. I'll list the most important routines that appears very often in my high level implementation of Lanczos. Then in the final python code, I'l just call these wrappers instead of using raw SQL again and again.

create_vector_or_matrix: declare a vector/matrix variable

assign_to: Copy a variable's value to another variable

vetorr_length: Return the length of a vector

vector_dot_product: take the dot product of two vectors

reverse_matrix: Append reverse of every edge into original graph

matrix_multiply_matrix_overwrite: Multiply a matrix with a matrix

matrix_multiplt_vector_overwrite: Multiply a matrix with a vector

normalzed_vector: Normalize a vector

3.5.3 SQL code

Please refer to the code in **Appendix**.

3.6 Belief Propagation

For belief propagation, we use the fabp method proposed in paper[8].

3.6.1 method

It can be shown that the solution of belief propagation can be approximated by the linear system:

$$[\mathbf{I} + a\mathbf{D} - c\mathbf{A}]\mathbf{b_h} = \phi_\mathbf{h}$$

where $\mathbf{A}$ is the n by n symmetric adjacency matrix, $\mathbf{D}$ is the diagonal matrix of degrees, b_h corresponds to the vector of final beliefs for each node, ϕ_h is prior belief vector, and h_h is the homophily factor, $a = 4h_h^2/(1 - h_h^2)$ and $c = 2h_h/(1 - 4h_h^2)$.

To solve this linear system, we can see :$\mathbf{I} + a\mathbf{D} - c\mathbf{A}$ as the form $\mathbf{I} - \mathbf{W}$, where $\mathbf{W} = -a\mathbf{D} + c\mathbf{A}$, and using the expansion:

$$(\mathbf{I} - \mathbf{W})^{-1} = \mathbf{I} + \mathbf{W} + \mathbf{W}^2 + \mathbf{W}^3 + ...$$

and the solution of the linear system is given by the formula:

$$\mathbf{b_h} = (\mathbf{I} - \mathbf{W}^{-1})\phi_\mathbf{h} = \phi_\mathbf{h} + \phi_\mathbf{h}\mathbf{W} + \phi_\mathbf{h}\mathbf{W}^2 + \phi_\mathbf{h}\mathbf{W}^3 + ...$$

Given this power method, the implementation is pretty straightforward as described in algorithm 9.

Algorithm 9 Belief Propagation

Bulk load graph into an edge table in database.
Initialize $h_h = 0.001$
Initialize the initial belief of every node as prior belief
repeat
  Update the belief of node by $b_h(i) = b_h(i-1)\mathbf{W} + \phi_{\mathbf{h}}$
until Convergence

3.6.2 Idea of SQL Implementation

The major computation involved is matrix vector multiplication, which is easy to implement in SQL using join and group by step. Furthermore we've wrapped all the matrix related operation in our host language Python, as described in previous task.

3.6.3 SQL code

Please refer to the code in **Appendix**.

3.7 Count of Triangle

We use a simple technique proposed by [14], its general idea is build upon a theorem that the count of triangles in a graph is proportional to the sum of cubes of eigenvalues of the graph.

3.7.1 Global triangle

The algorithm to calculate global triangles is as follows:

Math The formula to count global triangle is sum of cubes of eigenvalues, which is:

$$\Delta(G) \leftarrow \frac{1}{6} \sum_{j=1}^{i-1} \lambda_j^3 \tag{6}$$

3.7.2 Local triangle

The algorithm to calculate the local triangle is as follows.

Math Δ_i is the number of local triangles that node i participated in. The formula of local triangle count is based on the following theorem:

$$\Delta_j = \frac{\sum_{k=1}^{i-1} u_{jk}^2 \lambda_k^3}{2} \tag{7}$$

Algorithm 10 The EigenTriangle algorithm

Require: Adjacency matrix A (n X n)
Require: Tolerance *tol*
Output: $\triangle'(G)$ global triangle estimation

$\lambda_i \leftarrow LanczosMethod(A, 1)$
$\overrightarrow{\Lambda} \leftarrow [\lambda_1]$
$i \leftarrow 2$\{ initialize i, $\overrightarrow{\Lambda}$ \}
repeat
    $\lambda_i \leftarrow LanczosMethod(A, i)$
    $\overrightarrow{\Lambda} \leftarrow [\overrightarrow{\Lambda}\lambda_i]$
    $i \leftarrow i + 1$
until $0 \leq \frac{|\lambda_i^3|}{\sum_{j=1}^{i} |\lambda_i|^3} \leq$ tol
$\triangle'(G) \leftarrow \frac{1}{6}\sum_{j=1}^{i} \lambda_i^3$
return $\triangle'(G)$

Algorithm 11 The local eigentriangle algorithm[14]

Require: Adjacency matrix $A(n \times n)$
Require: Tolerance *tol*
OUTPUT: $\Delta'(G)$ per node triangle estimation
$\langle \lambda_1, \vec{u_1} \rangle \leftarrow LanczosMethod(A, 1)$
$\vec{\Lambda} \leftarrow [\lambda_1]$
$\bigcup \leftarrow [\vec{u_1}]$
$i \leftarrow 2$
repeat
    $\langle \lambda_i, \vec{u_i} \rangle \leftarrow LanczosMethod(A, i)$
    $\vec{\Lambda} \leftarrow [\vec{\Lambda}\lambda_i]$
    $\bigcup \leftarrow [\bigcup \vec{u_1}]$
    $i \leftarrow i + 1$
until $0 \leq \frac{|\lambda_i^3|}{\Sigma_{j=1}^{i-1}\lambda_j^3} \leq tol$
for $j = 1$ to n **do**
    $\triangle_j = \frac{\Sigma_{k=1}^{i-1} u_{jk}^2 \lambda_k^3}{2}$
end for
$\triangle(G) \leftarrow [\triangle_1, \ldots, \triangle_n]$
return $\triangle(G)$

3.7.3 Idea of SQL implementation

We will call Lanczos to get the eigenvalues and eigenvectors of the graph, and sum up the number using SQL *select*.

3.7.4 SQL code

Please refer to the code in **Appendix**.

3.8 Shortest Path

We adopt Dijkstra's shortest algorithm to compute shortest path from the source node. It works for directed weighted graph which has $O(|V|log|V|+|E|)$ time complexity. The pseudo code is listed in algorithm 3.8.1.

3.8.1 Math

The core idea of Dijkstra's idea is that it greedily select a candidate node which has already obtained its minimum distance, then update its neighbors' current best distance.

Algorithm 12 Dijkstra shortest path algorithm

Input: Source node and directed weighted graph.
Output: Shortest path of every node from source node.

```
for all node V in graph do
    dist[V] ← infinity
    visited[V] ← false
end for
dist[source] ← 0
insert source into Q
while Q is not empty do
    u ← vertex in Q with smallest distance
    remove u from Q
    visited[u] ← true
    for all neighbour v of u do
        alt ← dist[u] + dist_between(u, v)
        if alt < dist[v] && !visited[v] then
            dist[v] ← alt
            insert v into Q
        end if
    end for
end while
```

3.8.2 Idea of SQL implementation

This is implemented purely in SQL. Just like ordinary C code, I have a table to record the current status(visited, distance) of each node, and update one node at a time.

3.8.3 SQL code

Please refer to the code in **Appendix**.

3.9 Minimum Spanning Tree

We use the classical Prim's algorithm for this additional task. We abandon the Kruskal's algorithm because the disjoint set is not easy to implement using SQL.

3.9.1 method

The general idea of Prim's algorithm is to first initialize a tree with a single vertex, chosen arbitrarily from the graph, Then we grow the tree by one edge, the one that connect the tree to vertices not yet in the tree with minimum cost(weight). We repeat this process until all of the nodes in a graph is in the tree(of course we're assuming that the graph is weighted and connected).

3.9.2 Idea of SQL implementation

The implementation in SQL can be described in algorithm 3.9.2.

Algorithm 13 Prim's algorithm

Input: Edge Table E of a undirected connected graph
output: Edge Table MST containing edges of the minimum spanning tree

```
Create a node table N
Randomly insert a node into N
for i = 1 to numberofnodes − 1 do
    Insert into MST an edge from E with minimum weight where src node is in N and
    destination node is not in N
    Insert into N with the destination node of the edge selected in last step
end for
```

3.9.3 SQL code

Please refer to the code in **Appendix**.

4 Experiments

4.1 Dataset

We have conducted experiments on the following datasets(Mainly For Phase 1):

Dataset Name	Advogato
Largest conn compo	5054
Size	6551 vertices
Volume	51332 edges

In out final experiment suit, a more large and diverse datasets from different domains will be explored. The tentative source of experiment dataset are listed below:

SNAP: It has abundant data about social network, we plan to conduct triangle counting, pagerank, radius computing experiment which will reveal the underlying feature of large graphs and spot strange graphs.

Konect: Konect has more diverse datasets compared to SNAP, like citation network. It's a good target to analyze features of non-social networks, we will examine whether such networks follow power law by generating degree distribution, etc.

We also calculated statistics about the weakly connected components in table **??**.We compute radius for every node in Advogato dataset, the result is in table 1:

4.2 Task 1: Degree distribution

4.2.1 Description

We conduct 5 experiments on large scale graph data(all with more than 1 million nodes). The details of each dataset is as follows:

name	nodes	edges
Roadnet-ca	1,965,206	5,533,214
Roadnet-PA	1,088,092	3,083,796
Roadnet-TX	1,379,917	3,843,320
wiki-Talk	2,394,385	5,021,410
Youtube	1,134,890	2,987,624

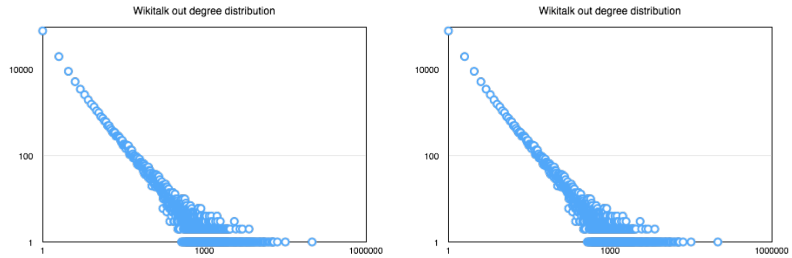
4.2.2 Detailed Plots

Following are the rank-frequency plots of each dataset, (a)(b) shows the in degree and out degree out Wikitalk. (c) is Roadnet-Ca. (d) is Roadnet-PA. (e) is Roadnet-TX. (f) is Youtube.

Proof of Correctness: For each dataset, we compare the plot with its original ground truth plot if it has.[1]. They are almost the same.

[1]Stanford SNAP does not have degree distribution, KONECT has degress distribution

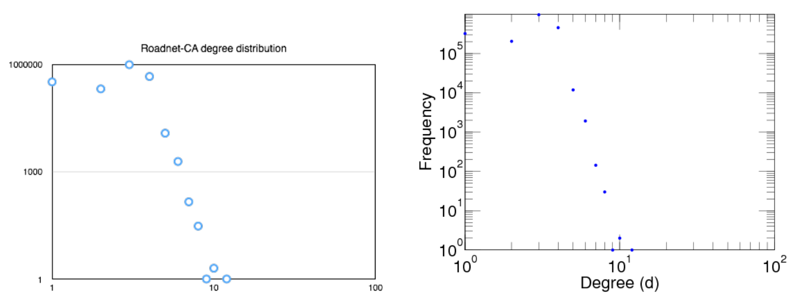
Wiki-Talk In the wiki-talk graph, it's directed. Each node represents a user, and an edge represents a user at least edited another user's talk page. Figure 1 shows the in and out degree of this dataset. We don't have official degree distribution plot for this dataset. We can observe **power law** from this plot.



(a) (b)

Figure 1: In degree distribution (a) and out degree distribution (b) of Wikitalk.

Roadnet-CA Roadnet-CA is un undirected graph. Each node represents an intersection of road and edge is the road segments between two road intersections. Figure 2(a) shows our plot of degree distribution, Figure 2(b) is the original degree distribution of the dataset. We can see that they are the same. There is no obvious power law.

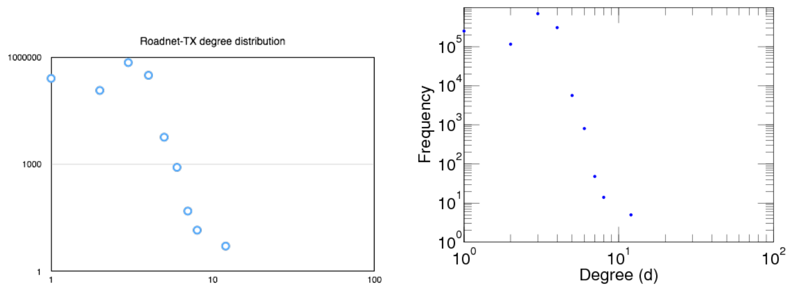


(a) (b)

Figure 2: (a)Roadnet-CA (b) official degree distribution of Roadnet-CA

Roadnet-TX Roadnet-TX is an undirected graph. Each node represents an intersection of road and edge is the road segments between two road intersections. Figure 3(a) shows our plot of degree distribution, Figure 3(b) is the original degree distribution of the dataset. We can see that they are the same. There is no obvious power law.

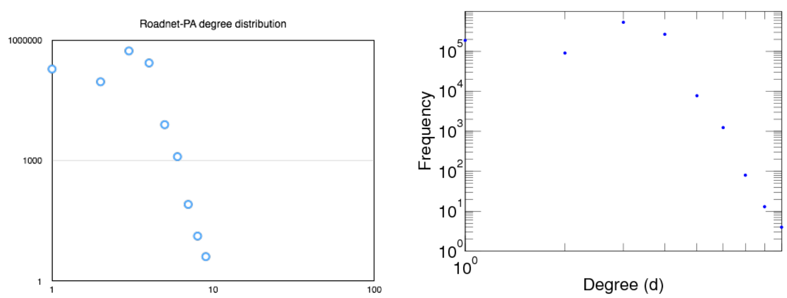
Roadnet-PA Roadnet-pa is un undirected graph. Each node represents an intersection of road and edge is the road segments between two road intersections. Figure 4(a) shows our



(a) (b)

Figure 3: (a)Roadnet-TX (b) official degree distribution of Roadnet-TX

plot of degree distribution, Figure 4(b) is the original degree distribution of the dataset. We can see that they are the same. There is no obvious power law.



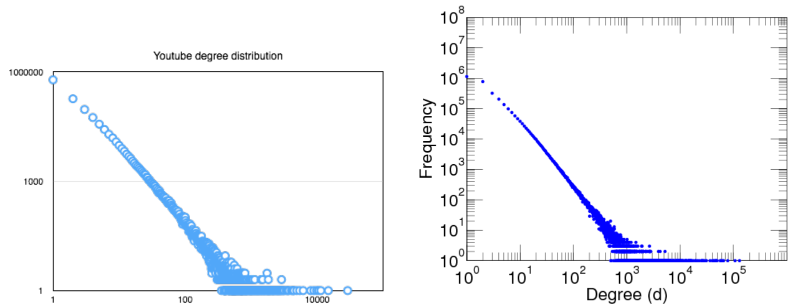
(a) (b)

Figure 4: (a)Roadnet-pa (b) official degree distribution of Roadnet-pa

Youtube Youtube is an undirected graph, each node is a user in youtube.com, an edge represents they are friends. Figure 5(a) shows our plot of the dataset, Figure 5(b) is the official degree distribution of the dataset. We can see that they are exactly the same and there exhibits **power law**.

4.2.3 Observation

As we can observe, that most *social network* exhibit perfect power law in degree distribution. It aligned with our intuition. While for the series of Roadnet dataset, we can not observe obvious power law. So maybe we can conclude that not all graphs have power law property in it. Some datasets like roadnet, involves a lot of human design, is less *chaotic* than ordinary graphs.



(a) (b)

Figure 5: (a)Youtube (b) official degree distribution of Youtube

4.3 Task 2: Pagerank

In this experiment, we run pagerank on various size of graph data, 10 in total. The purpose of this design is that we think pagerank is a time consuming operation, and our first try proves our intuition. So we want to test the implementation on incrementally larger dataset, so that we can spot the bottleneck of single machine implementation.

Name	Nodes	edges
Roadnet-ca	1,965,206	5,533,214
Roadnet-PA	1,088,092	3,083,796
Roadnet-TX	1,379,917	3,843,320
com-Amazon	334,863	925,872
web-BerkStan	685,230	7,600,595
email-Enron	36,692	367,662
web-Google	875,713	5,105,039
soc-Slashdot0902	77,360	905,468
web-Stanford	281,903	2,312,497
wiki-Vote	7,115	103,689

4.3.1 Detailed Plot

Proof of Correctness: In order to verify the correctness of our implementation, we compare the SQL implementation with a standard implementation with NumPy[2] on a tiny dataset from "Introduction to Information Retrieval". Table 2 is the outcome of two different implementaions. We can see that most of the values are the same to one digit of the right of decimal point. The difference after that is due to subtle difference in implmentation, such as max iteration, initilization, etc.

Here We use rank-frequency plot to find the underlying patterns in the datasets.

[2]http://www.numpy.org/

Node	SQL	NumPy
1	0.122864	0.11799887
2	0.081234	0.08069813
3	0.263843	0.2526222
4	0.529185	0.52647168
5	0.452971	0.45487423
6	0.081234	0.08069813
7	0.645657	0.65203598

Table 2: SQL implementaion VS NumPy implementaion

Roadnet-CA The rank-frequency plot(Figure 6) of Roadnet-CA is nearly flat, there is not obvious pattern in the plot.

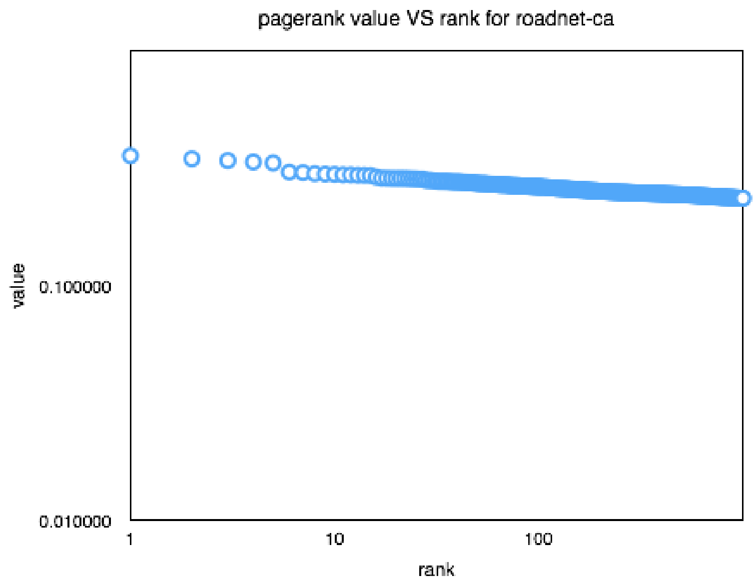


Figure 6: Rank-frequency plot Roadnet-CA

Roadnet-TX The rank-frequency plot(Figure 7) of Roadnet-TX is nearly flat, there is not obvious pattern in the plot.

Roadnet-PA The rank-frequency plot(Figure 8) of Roadnet-PA is nearly flat, there is not obvious pattern in the plot.

com-Amazon From the plot(Figure 9) we can find that the pagerank distribution follows **power law**, which means that top few nodes has the largest pagerank, while the majority

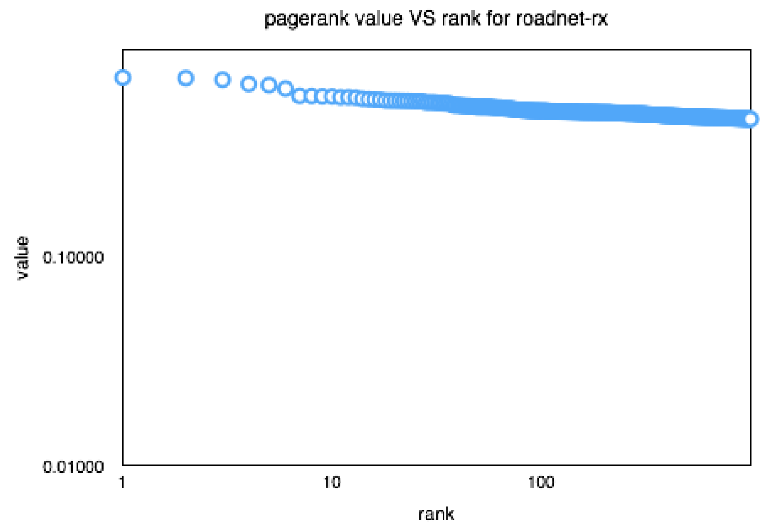


Figure 7: Rank-frequency plot Roadnet-TX

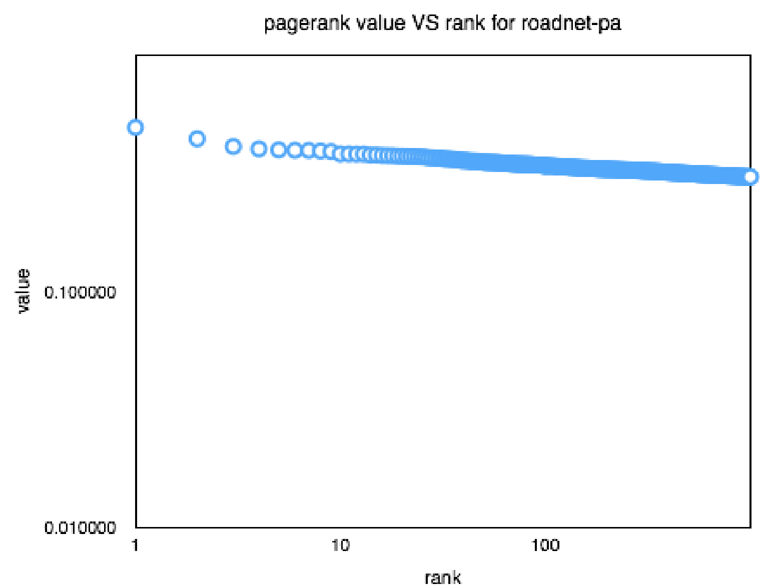


Figure 8: Rank-frequency plot Roadnet-PA

of the node has small pagerank. This matches out intuition in real world data.

web-BerkStan From the plot(Figure 10) we can find that the pagerank distribution follows **power law**(exception the first few nodes), which means that top few nodes has the largest pagerank, while the majority of the node has small pagerank. This matches out intuition in real world data.

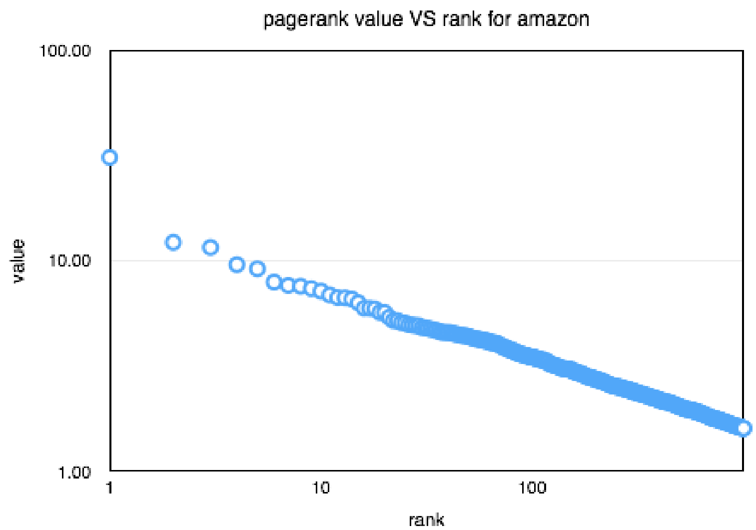


Figure 9: Rank-frequency plot Amazon

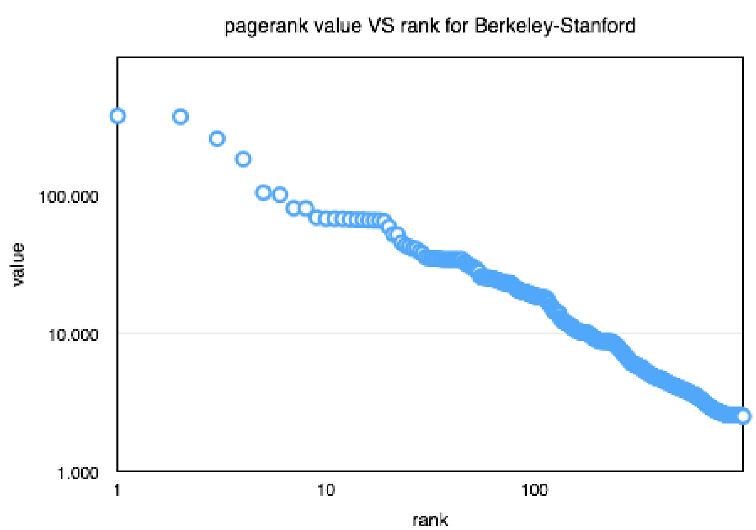


Figure 10: Rank-frequency plot BerkeStan

email-Enron From the plot(Figure 11) we can find that the pagerank distribution follows **power law**(except the first few nodes), which means that top few nodes has the largest pagerank, while the majority of the node has small pagerank. This matches out intuition in real world data.

web-Google From the plot(Figure 12) we can find that the pagerank distribution follows **power law**, which means that top few nodes has the largest pagerank, while the majority of the node has small pagerank. This matches out intuition in real world data.

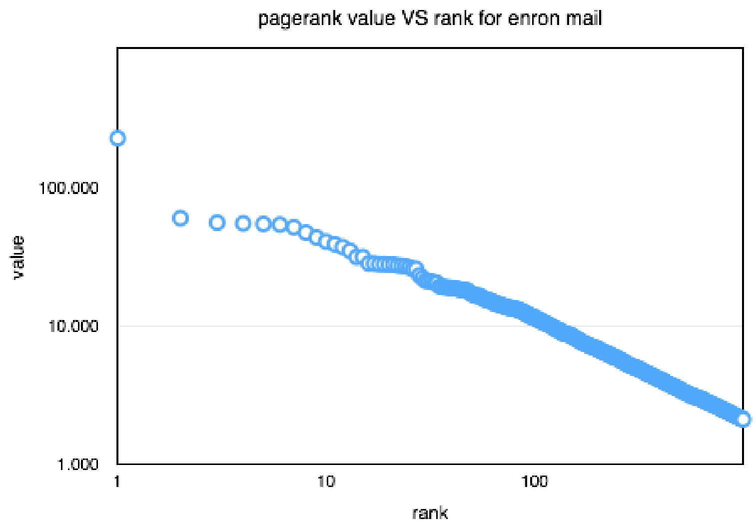


Figure 11: Rank-frequency plot Enron mail

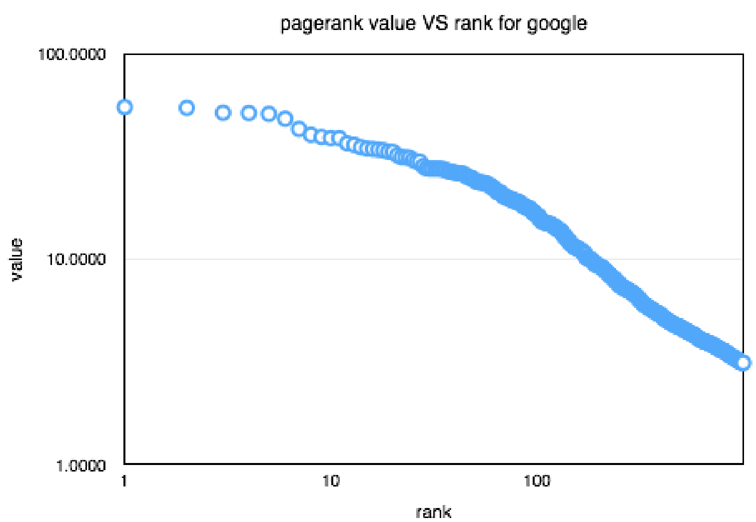


Figure 12: Rank-frequency plot Google

soc-Slashdot0902 From the plot(Figure 13) we can find that the pagerank distribution follows **power law**, which means that top few nodes has the largest pagerank, while the majority of the node has small pagerank. This matches out intuition in real world data.

web-Stanford From the plot(Figure 14) we can find that the pagerank distribution follows **power law**(except the first few nodes), which means that top few nodes has the largest pagerank, while the majority of the node has small pagerank. This matches out intuition in real world data.

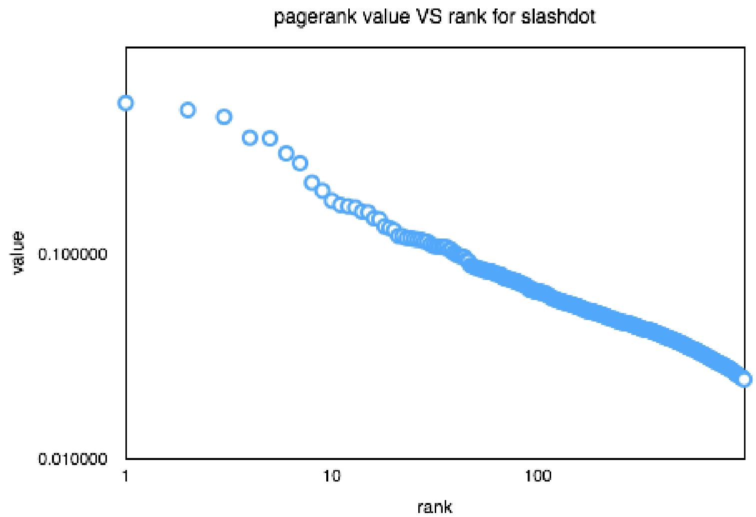


Figure 13: Rank-frequency plot Slashdot

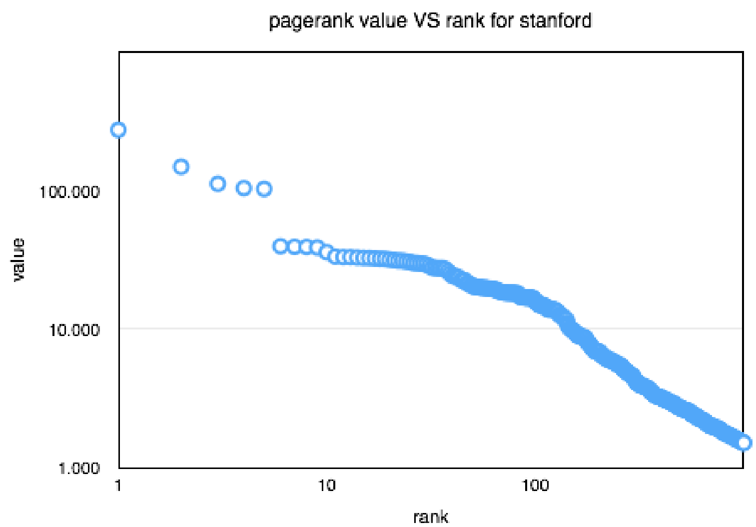


Figure 14: Rank-frequency plot Stanford

wiki-Vote From the plot(Figure 15) we can find that the pagerank distribution follows **power law**, which means that top few nodes has the largest pagerank, while the majority of the node has small pagerank. This matches out intuition in real world data.

4.3.2 Observation

We can observe that in log-log scale, all datasets exhibits linear relationship between rank and frequency. Thus we find another appearance of **power law** in natural graph. An abnormal phenomenon is that the slope of Roadnet datasets is nearly flat. This reflects

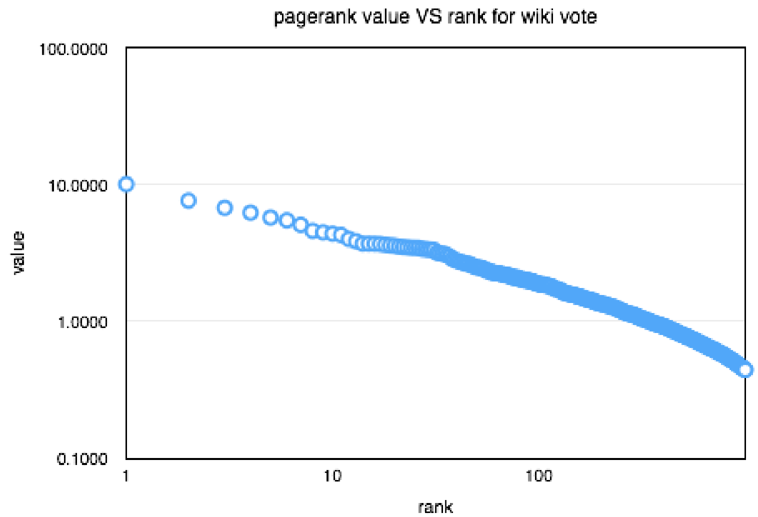


Figure 15: Rank-frequency plot Wiki-Vite

another aspects of the fundamental property of Roadnet, that all nodes in a roadnet graph are nearly equal to each other, it's a decentralized graph. While in ordinary social network, we always expect to have some important authorities.

4.4 Task 3: Weakly Connected Component

4.4.1 Validity

We verify the validity of our algorithm in both small and large datasets. For small datasets, we generate several synthetic graph containing a small amount of weakly connected components of different sizes. Then we run our algorithm on this small datasets, the result matches the reality. For large datasets, we use real graph from Konect and SNAP project, we can reproduce the statistics "number of nodes in largest WCC" on their web using our algorithm, which partially demonstrate the correctness of our algorithm .

4.4.2 Experiment on large datasets

In this experiment, we run weakly connected component in several datasets of various size (from 100 thousand to 2 million of nodes). In the following, we show the frequency radius plot in log log scale for dataset Email-EuALL (figure 16), Soc-Sign-Epinions(figure 17,) Trec-wt10g (figure 18), Google Web Graph (figure 19), Wiki Talk (figure 20).

We also run our algorithm in two large connected graphs(com-Youtube and com-amazon), some statistics are presented in table 3

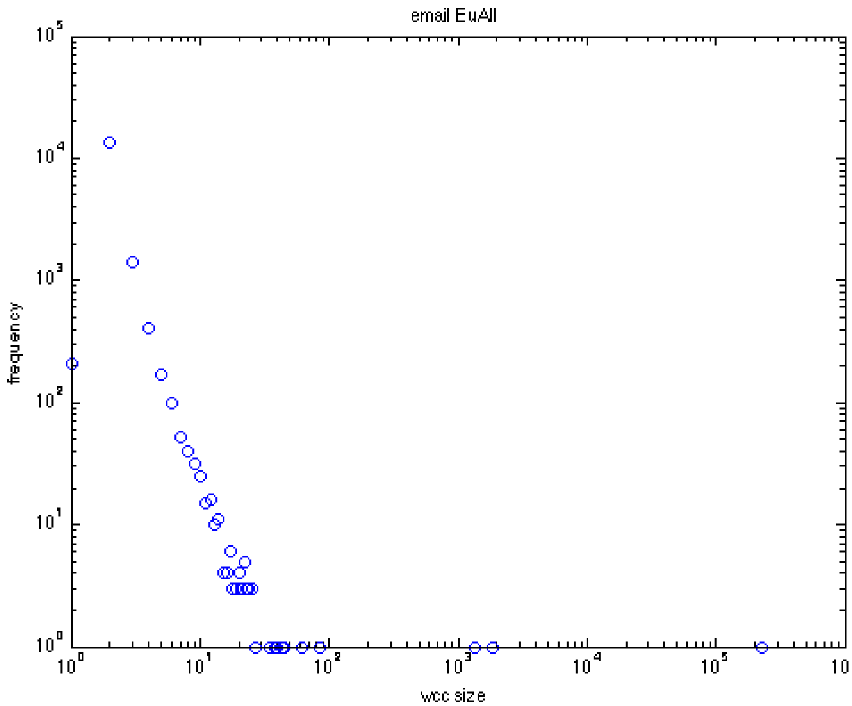


Figure 16: Email-EuAll

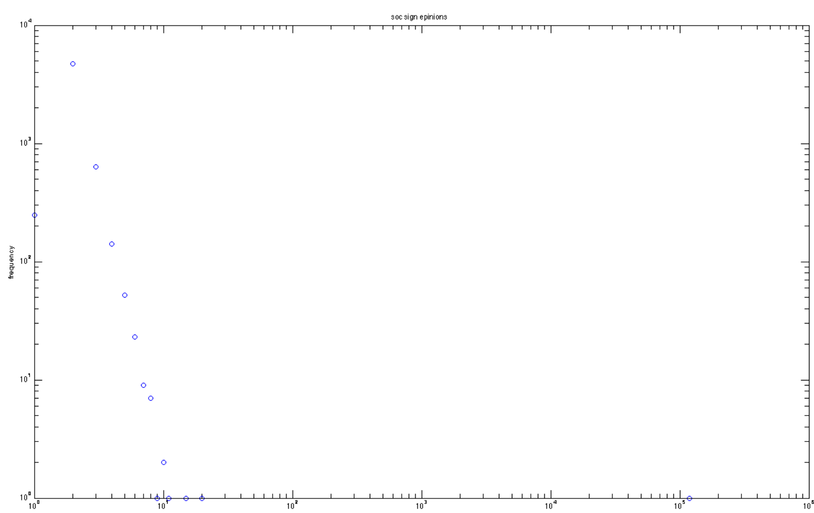


Figure 17: Soc-Sign-Epinions

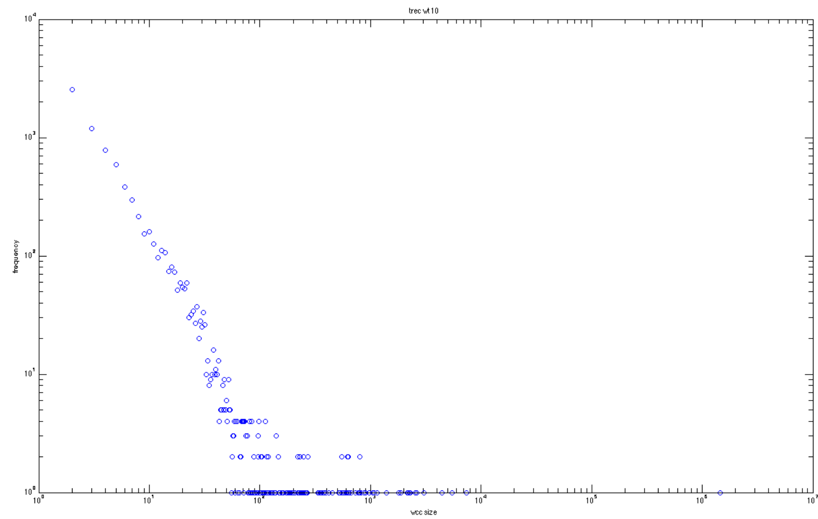


Figure 18: Trec-wt10g

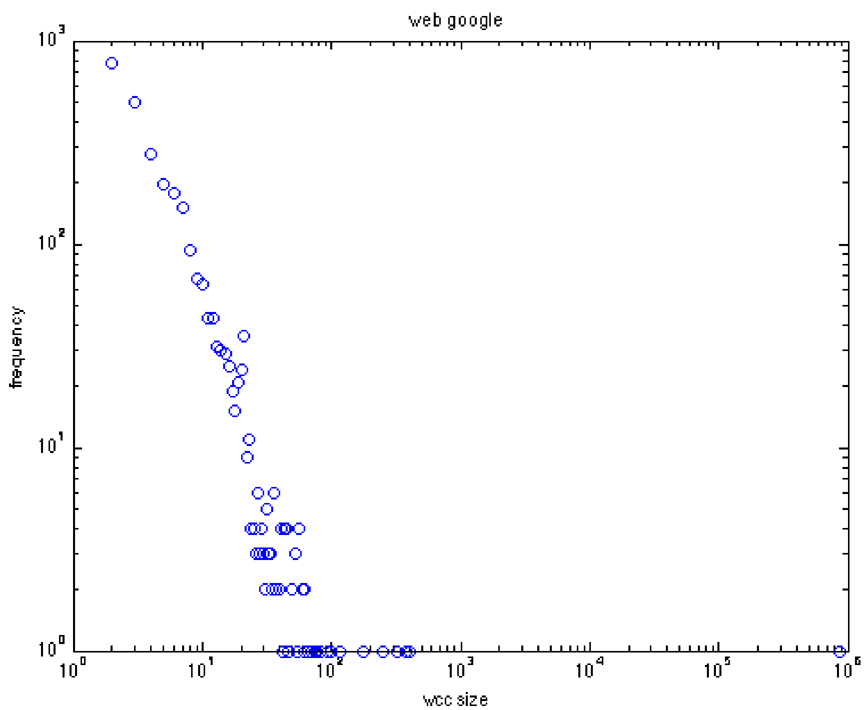


Figure 19: Google Web Graph

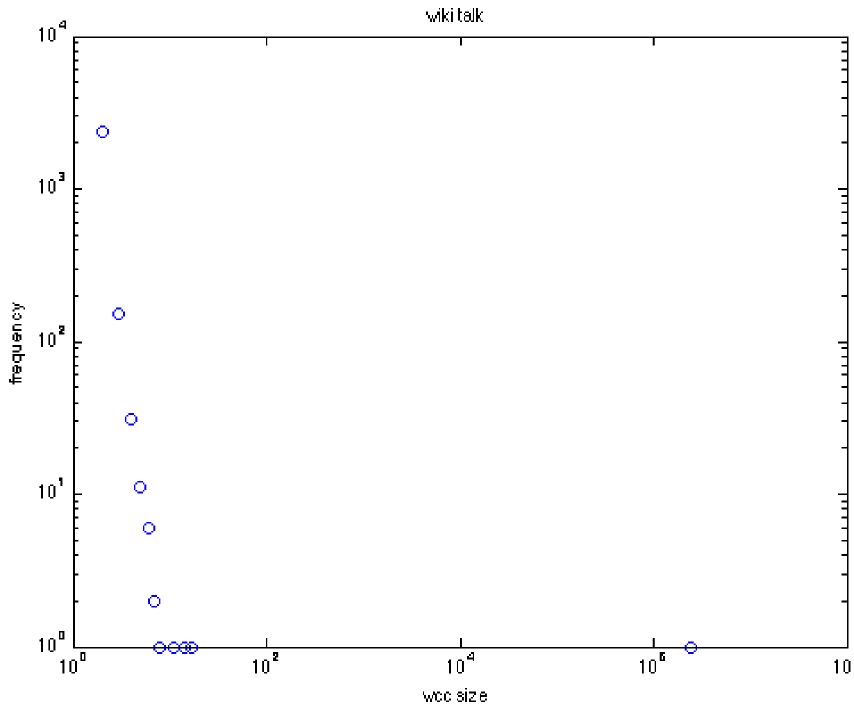


Figure 20: Wiki Talk

Table 3: Weakly Connected Component Run Result

graph	number of nodes	number of edges	number of wcc	nodes in largest wcc
com-Youtube	1134890	2987624	1	1134890
web-Google	875713	5105039	2746	855802
com-Amazon	334863	2987624	1	1134890
email-EuAll	265214	420045	15836	224832
wiki-talk	2394385	5021410	2555	2388953
soc-sign-epinions	131828	841372	5816	119130
trec wt10g	1601787	8063026	7955	1458316

4.4.3 Observation

1. From the log-log scale frequency-size plot, we find that generally frequency-size follows the power law. Connected components of small sizes tend to occur more often those of larger sizes.

2. In all the plots, there is a giant connected component that contains the majority of

nodes in the graph. This is a characteristic that almost all large graphs share, for example, in the yahoo web graph, there is also a giant connected component.

4.5 Task 4: Radius of Every Node

4.5.1 Experiment on large datasets

In this experiment, we run our radius algorithm in several large datasets. The statistics of these datasets are presented in Table 4

Table 4: Datasets Statistics

dataset	number of vertices	number of edges	diameter
DBLP co-authorship network	317080	1049866	21
Epinions social network	131828	841372	14
Amazon product co-purchasing network	334863	925872	44
EU email communication network	265214	420045	14
Google web graph	875713	5105039	21
Youtube social network	1134890	2987624	20

4.5.2 Observation

1. Radius Distribution: From the above plot, we find radius distribution of these graphs either tend to be single-modal, for example, EU Email Communication graph in figure 21 and Epinions Social Network in figure 22, or bi-modal like DBLP co-authorship network in figure 23 and Youtube Social Network in figure 25.

2. Relationship to the connected component: So why radius tend to be distributed like this, we guess it's somehow related to the connectivity of the graph. Therefore we take a look back to the statistics we got from task3. We find that the graphs that has a single-modal shape radius distribution are fully or almost fully connected. The graphs that has bi-modal radius distribution are not that well connected, For most nodes that appear in the Giant Connected Component(GCC), they tend to have a higher radius value, specifically the smaller radius value it has, the more centric it is in the GCC. On the other hand, The first peak in the radius distribution represent those disconnected components.

4.5.3 Proof of Correctness

Since the algorithm we use is an approximate algorithm, it's hard for us to verify the validity of our algorithm accurately. But we still try the algorithm on both small, large and synthetic datasets to demonstrate its validity. First we test the algorithm on the synthetic tiny dataset consisting of only 10 nodes and 8 edges. It accurately compute the radius of every node.

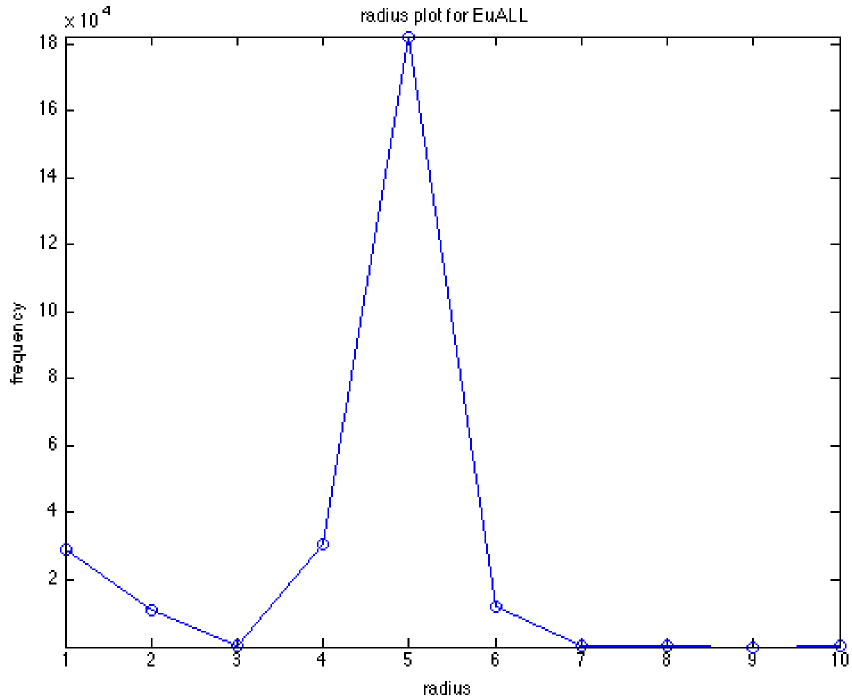


Figure 21: EU Email Communication

Then we test our algorithm on small datasets of several thousands of nodes, the experiment result are showed in Table 5, we can see that our algorithm can get a closer estimate of diameter.

Table 5: Radius Experiment on Small Datasets

dataset	nodes	edges	real diameter	estimated diameter
Enron email network	36692	183831	11	11
High Energy Physics	12008	118521	13	12
Advogato Trust Network	6551	51332	9	9

For large dataset we run in last section, the estimated diameter is still bounded by real diameter, but the estimation is no longer that close. This can be explained by the approximation nature of the algorithm we use. The Flajolet-Martin string we generate for every node is not unique, and as the number of nodes grows larger, the uniqueness of a node that a Flajolet-Martin string can represent is weakened. Therefore, for graph with larger size, the estimation might become less accurate.

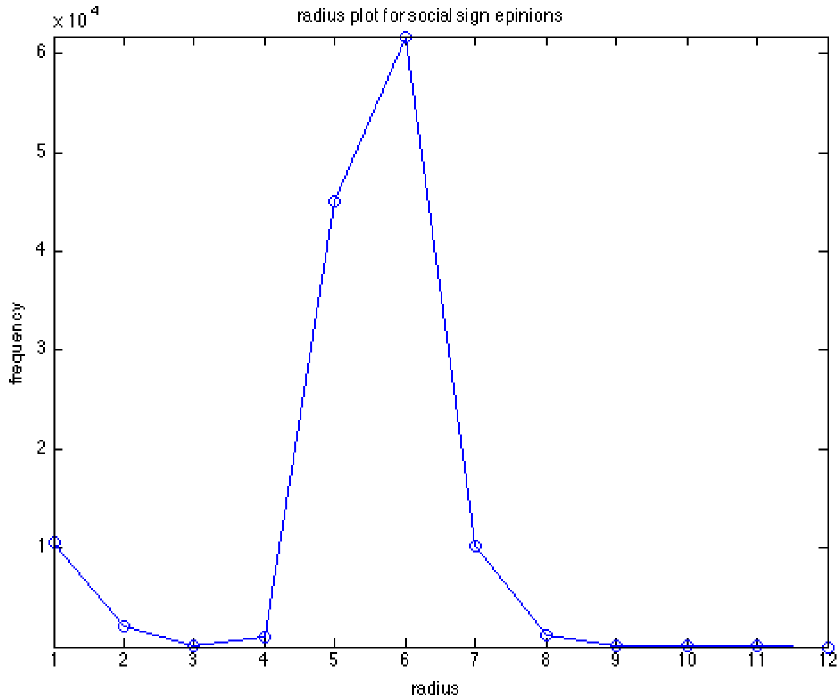


Figure 22: Epinions social network

4.6 Task 5: Eigenvalue/Singular value

In this experiment, we compute the eigenvalue of several datasets. The detailed description is in Table 6.

Name	Nodes	Edges
wiki-Vote	7,115	103,689
youtube	1,134,890	2,987,624
slashdot0922	82,168	948,464
com-DBLP	317,080	1,049,866
wiki-Talk	2,394,385	5,021,410

Table 6: Dataset for Task 5

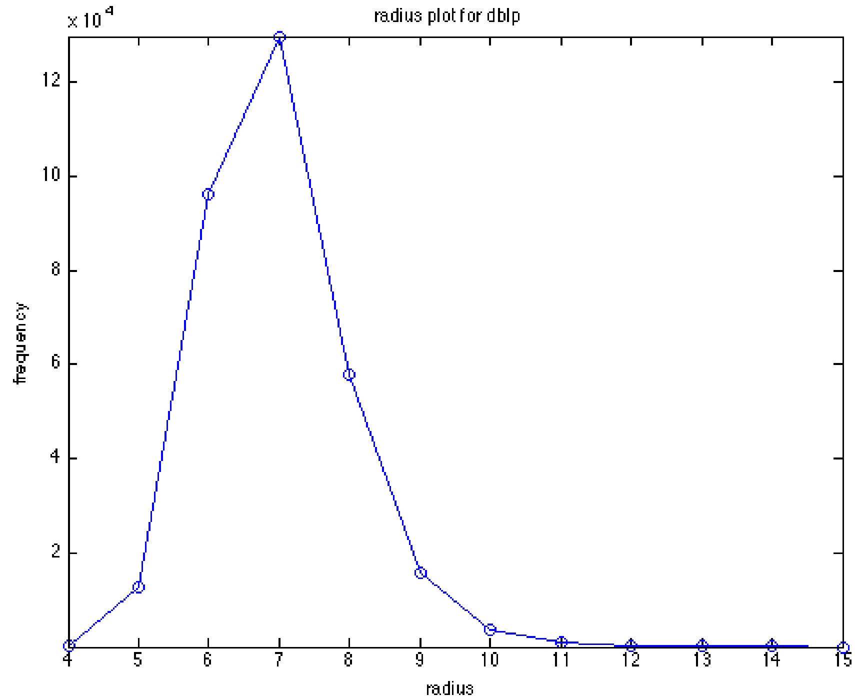


Figure 23: DBLP co-authorship network

4.6.1 Details

Proof of Correctness: In order to prove the correctness, we run our algorithm on the relatively smaller dataset, which is *Zachary karate club network*[3]. It has 34 nodes and 78 edges. The true top eigenvalues of Zachary is shown in Figure 27. We also calculated top-10 eigenvalues by our SQL implemention, the result is in Table 7. We can see that most of the top eigenvalues are near to its counter part in ground truth. So we are confident that our implementaiton is correct.

wiki-Vote The top 5 eigenvalues of wiki-Vote are 153.77, 107.23, 72.34, 49.23, -5.64. The plot(Figure 28) is as follows:

youtube The top 5 eigenvalues of Youtube are 238.55, 155.78, 83.29, 41.60, -7.5. The plot(Figure 29) is as follows:

slashdot The top 5 eigenvalues of slashdot are 125.34, 109.17, 85.23, 28.35, 4.34. The plot(Figure 30) is as follows:

[3]http://konect.uni-koblenz.de/networks/ucidata-zachary

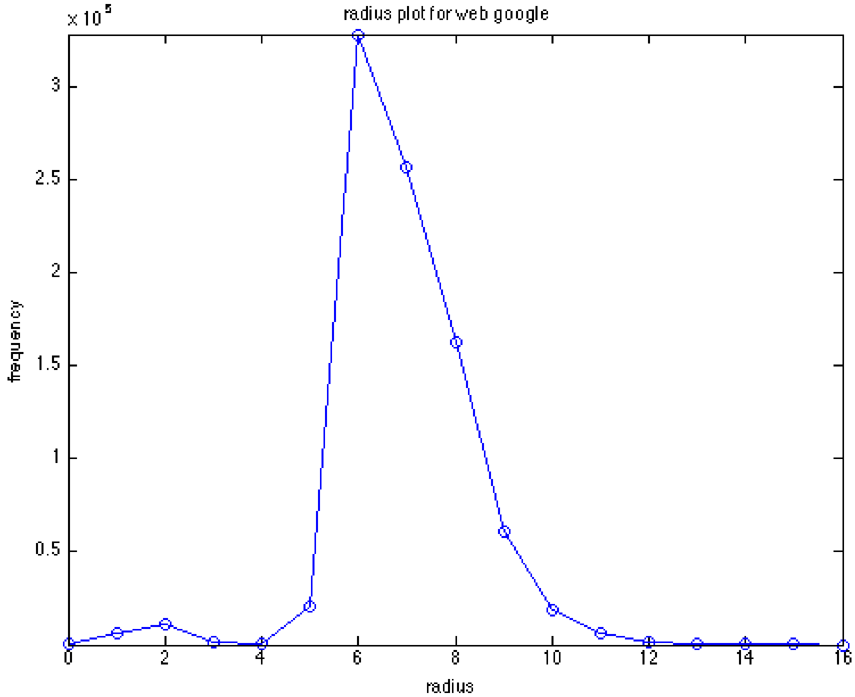


Figure 24: Google Web Graph

com-DBLP The top 5 eigenvalues of DBLP are 235.56, 173.45, 130.23, 95.14, 44.23. The plot(Figure 31) is as follows:

wiki-Talk The top 5 eigenvalues of Wiki-Talk are 353.23, 254.67, 125.82, 35.29, -33.45. The plot(Figure 32) is as follows:

4.6.2 Observation

We can see that in most datasets, the top eigenvalues drops quickly. This is also a reason why in Task 7, we only need top eigenvalues to count the number of triangles in a graph. Since the top eigenvalues is enough for the majority of the energy of the graph.

4.7 Task 6: Belief Propagation

4.7.1 Experiment on large datasets

In this experiment, we run our radius algorithm in several large datasets. The statistics of these datasets are presented in Table 8

Similar to semi-supervised learning, belief propagation algorithm require the graph to be partially labeled. However, we don't have such information. Therefore, in this experiment,

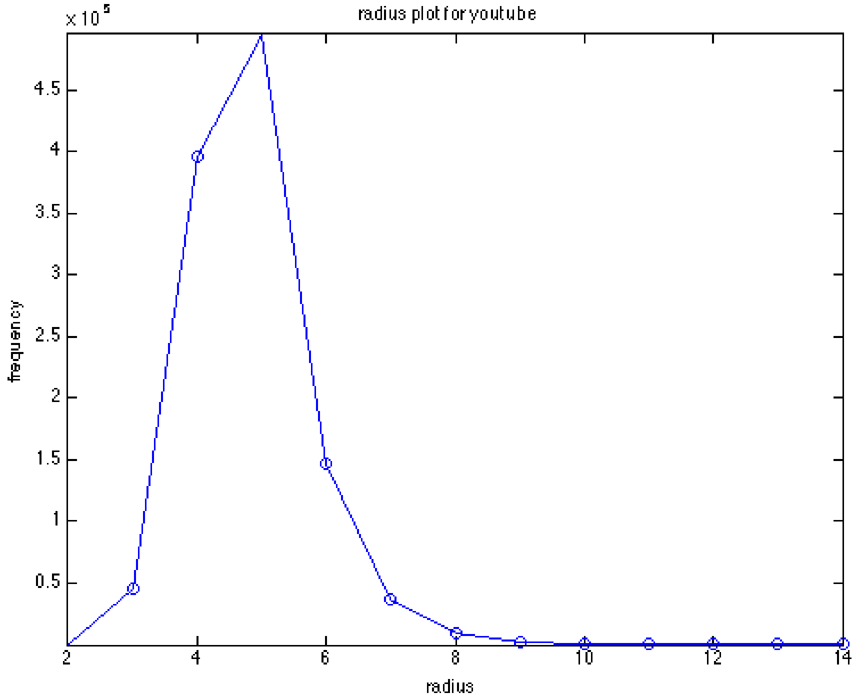


Figure 25: Youtube Social Network

we randomly assign the prior belief for all nodes. Specifically, we randomly assign 5% of the nodes with positive label, i.e. positive prior belief(0.001), and 5% of other nodes with negative label, i.e. negative prior belief(-0.001), and the rest with zero belief, meaning that we don't have prior knowledge for these nodes. Then we conduct FABP algorithm on these datasets, the result are shown in Table 9 .

4.7.2 Observation

1. By applying Belief Propagation algorithm on these graphs, most of the unlabeled nodes are successfully assigned either positive or negative belief.

2. We find that for some graphs, larger proportions of nodes gets labeled than other graph. For example, in DBLP co-authorship network and amazon product co-purchasing network, about 95% of the nodes get labeled using only 10% labeled nodes. While for like Epinions social network , only about 30% of the nodes get labeled. Once again, we look back at the connectivity of the graph to find the probable cause. We observed that, graph that is well connected is easier to get more nodes assigned with labels. This observation makes sense in that 'beliefs' can be easier to propagate in well connected graphs than those grape with many disconnected components.

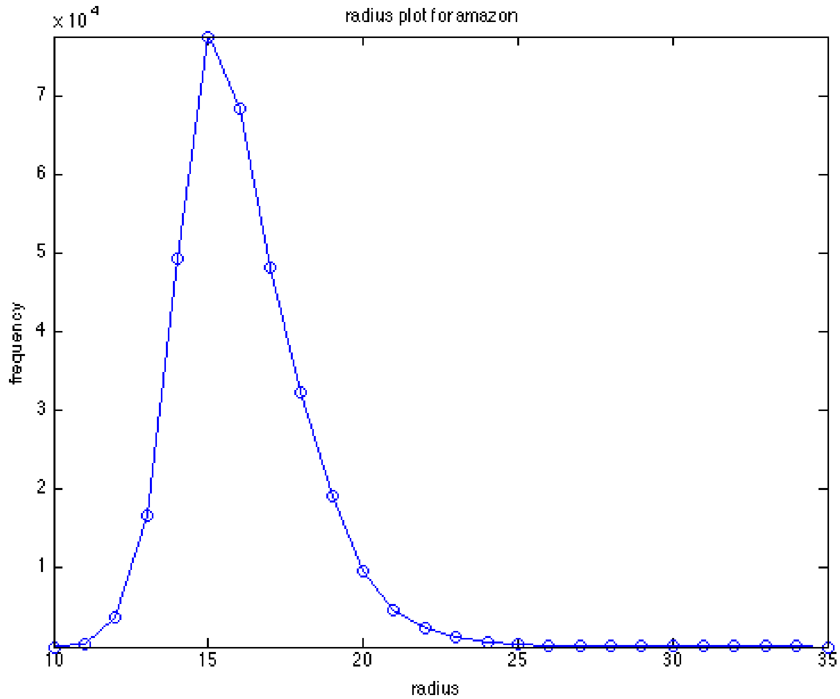


Figure 26: Amazon product co-purchasing networkk

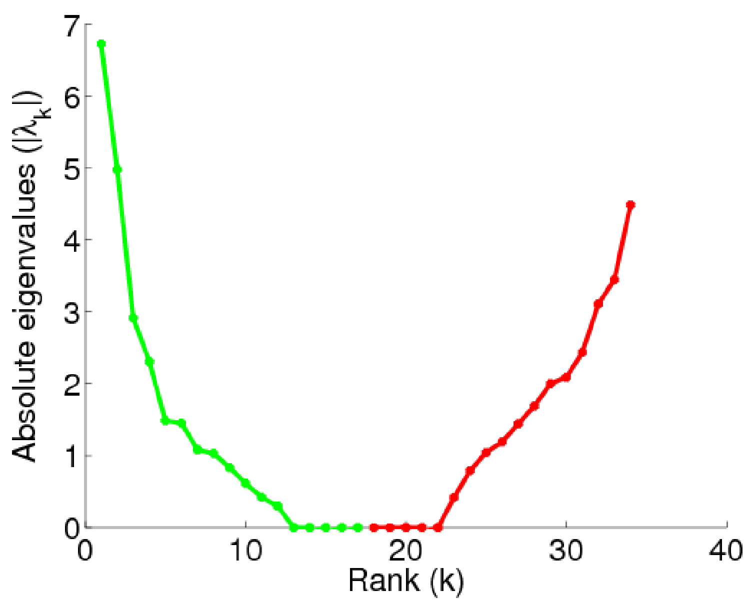


Figure 27: Top eigenvalues of Zachary

Rank	Eigenvalue
1	6.72569758513981
2	5.04694048054693
3	4.97613169036548
4	2.65295803734598
5	2.37853063223689
6	1.31746846315176
7	1.05904538243756
8	0.827084251502258
9	0.475203434096779
10	0.0182376036753603

Table 7: Top eigenvalues of Zachary

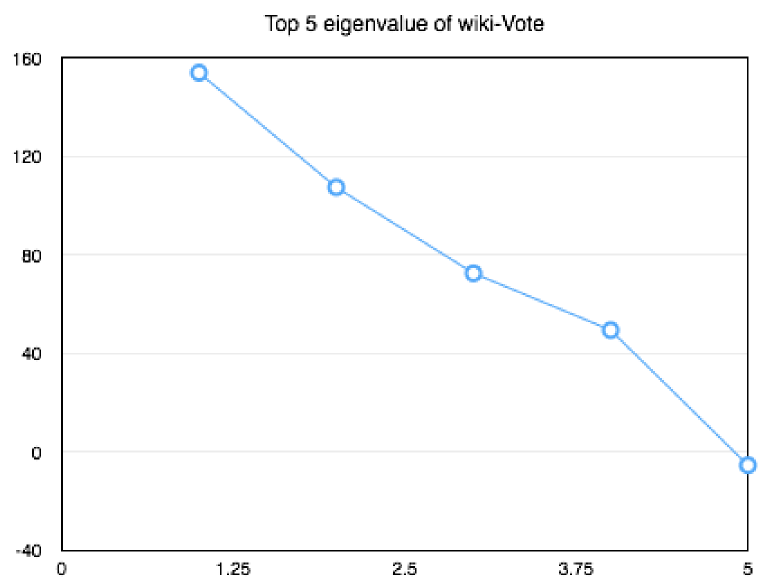


Figure 28: Top eigenvalues of wiki-Vote

4.7.3 Proof of Correctness

As mentioned in the previous section, we don't have any labels for these large datasets, therefore we verify the result according to statistics we got in the last section. We can see that the proportion with positive labels and negative labels are approximately same, which resembles the label distribution with our prior belief. Also, we successfully inferenced the belief of other nodes using only 10% labeled nodes. In order to verify the accuracy of our algorithm, we run FABP on small matrix we generate. In essence, the FABP tries to solve a linear system $(I - W)x = prior$, therefore we test our FABP against the linear system solver in MATLAB(function linsolve). And the two get nearly identical result(with error less than

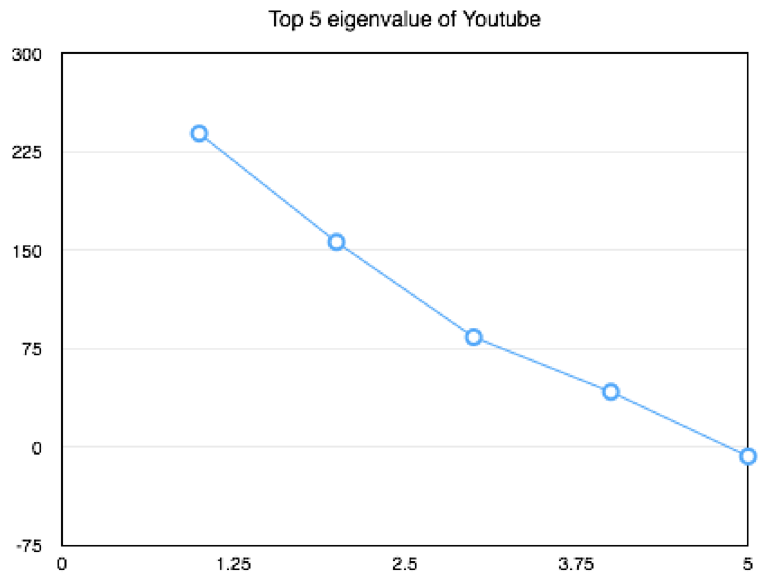


Figure 29: Top eigenvalues of Youtube

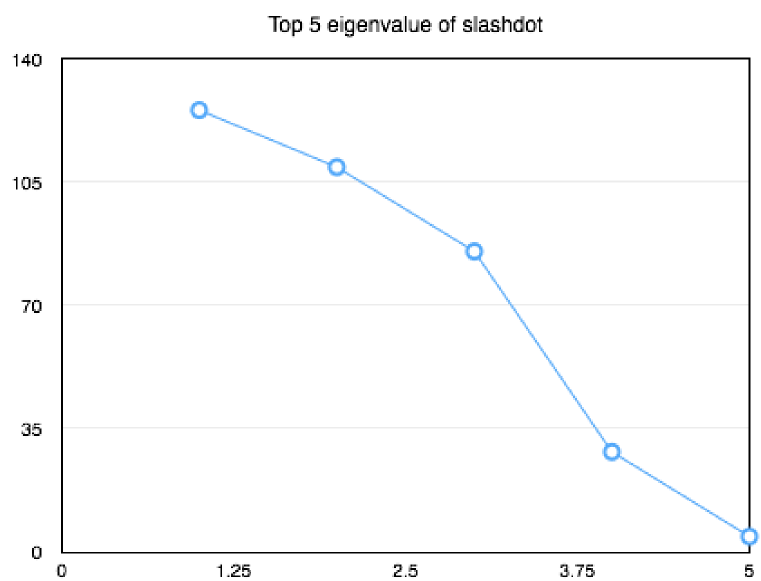


Figure 30: Top eigenvalues of Slashdot

0.01) in solving some toy linear systems consisting of 2 by 2 matrix. This demonstrates the correctness of our algorithm.

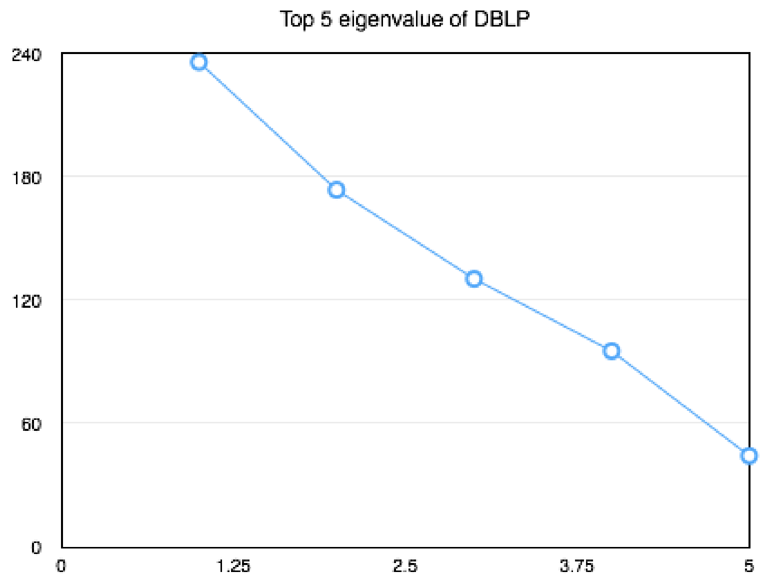


Figure 31: Top eigenvalues of DBLP

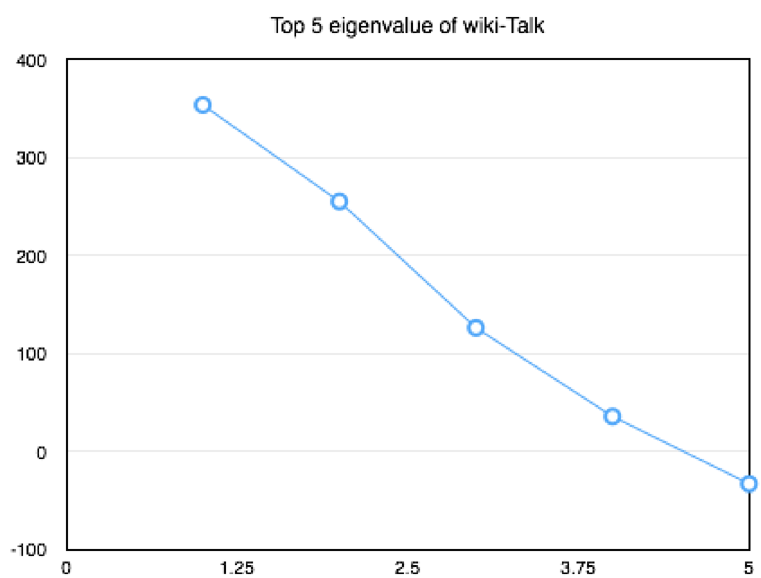


Figure 32: Top eigenvalues of wiki-Talk

4.8 Task 7: Triangle Counting

According to the algorithms in [14], we know that the number of triangles in a network is proportional to the sum of eigenvalue of its adjacency matrix, which is $\frac{\sum_i \lambda_i^3}{6}$. Figure 10 shows the running time of global triangle counting with regards to the size of graph. We can see that as the size of graph increases, the running time also grows nearly linearly with the

Table 8: Datasets Statistics

dataset	number of vertices	number of edges
DBLP co-authorship network	317080	1049866
Epinions social network	131828	841372
Amazon product co-purchasing network	334863	925872
EU email communication network	265214	420045
Google web graph	875713	5105039
Youtube social network	1134890	2987624

Table 9: BP Statistics

dataset	positively labeled	negatively labeled	unlabeled
DBLP co-authorship network	164103	144259	8718
Epinions social network	22334	22575	86919
Amazon product co-purchasing network	158837	160524	15502
EU email communication network	127481	109616	28117
Google web graph	385277	389975	100461
Youtube social network	584422	508628	41840

size. For the largest graph, which is Roadnet-PA, it runs nearly for an hour to complete. However, the predicted result for Roadnet-PA is unsatisfactory. We conduct both global and local triangle counting, all the data is listed as follows.

4.8.1 Detailed Plots

Proof of Correctness: In this experiment, we run our algorithm on the dataset, and verify that the predicted number of triangles is a good approximate of the true count. The full result is in Table 11. As we can see, most of the result is near to each other. So we are aure about the correctness of the implementation.

Table 10 lists run time of global triangle counting. Figure 33 plots the run time of global triangle counting on each dataset.

4.8.2 Local triangle counting

We plot the rank-frequency plot of local triangle counting, that x-axis represent the rank of the count of local triangle, y-axis represents the number of local triangles at that rank.

graph size	run time(seconds)
7115	45.199s
36692	72.76
82168	596.046
334863	1288.703
1088092	2980.985

Table 10: Task 7 run time(global)

dataset	size	predict	truth
wiki-Vite	7115	661282	608389
Enron-email	36692	756757	727044
slash-dot	82168	635792	602592
Amazon-com	334863	675778	667129
Roadnet-PA	1088092	72134	67150

Table 11: Predicted triangle count(global)

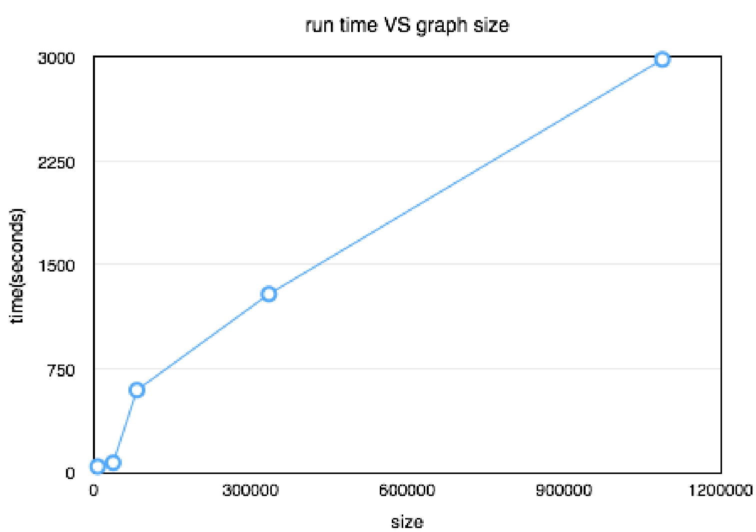


Figure 33: Task 7: Run time VS graph size(global)

Amazon Figure 34 plots the rank-frequency of Amazon. We can observe from the figure that it follows **power law**.

Enron mail Figure 35 plots the rank-frequency of Enron Mail. We can observe from the figure that it follows **power law**.

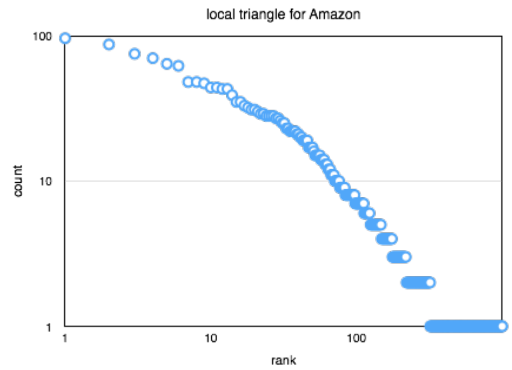


Figure 34: Local triangle counting for Amazon

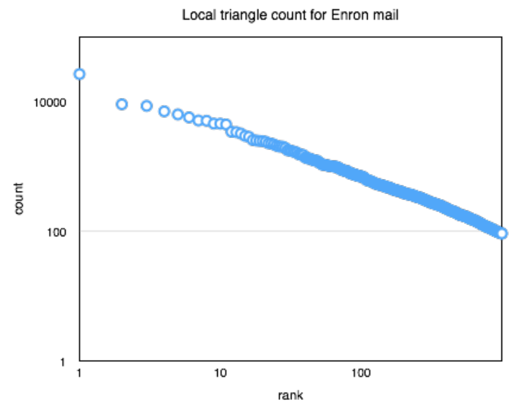


Figure 35: Local triangle counting for Enron mail

Slashdot Figure 36 plots the rank-frequency of Slashdot. We can observe from the figure that it follows **power law**.

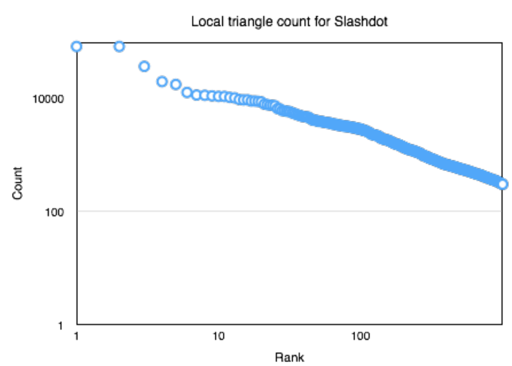


Figure 36: Local triangle counting for Slashdot

wiki-Vote Figure 37 plots the rank-frequency of wiki-Vote. We can observe from the figure that it follows **power law**.

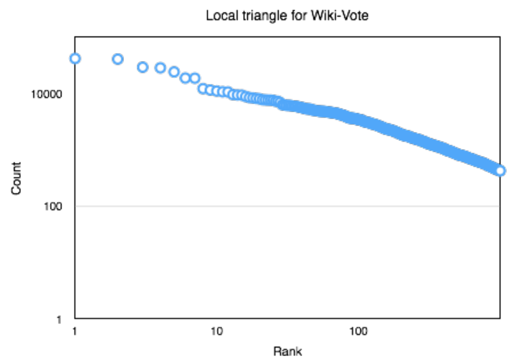


Figure 37: Local triangle counting for wiki-Vote

Youtube Figure 38 plots the rank-frequency of Youtube. We can observe from the figure that it follows **power law**.

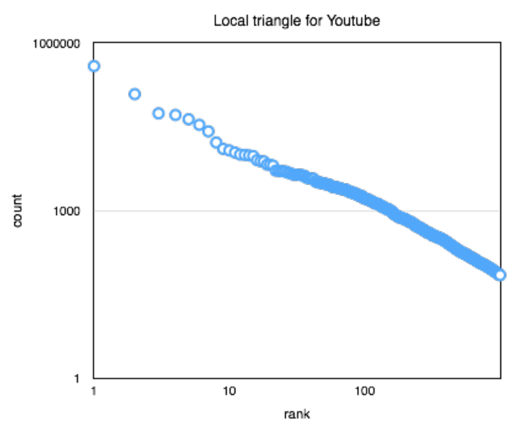


Figure 38: Local triangle counting for youtube

4.8.3 Observation

We can see that the rank-frequency plot of local triangle count also follows **power law**, it just matches our intuition. And we can observe that the run time grows nearly linearly with the graph size.

4.9 Innovative task: Shortest Path

Proof of Correctness: In order to prove the correctness, we conduct experiments on a small dataset that we manually constucted. And the result is exactly the same.

In order to test the performance of the SQL implementation, we run the shortest path algorithm on incrementally larger datasets, the running time erquired according the size of graph is ploted in Figure 39. We can observe from that figure that as the graph size increases, the running time grows much faster than linear. The reason for this is that we implemented all the data structures through tables, which incurs a lot of overhead of search. An alternative is to use the data types provided by Postgres.

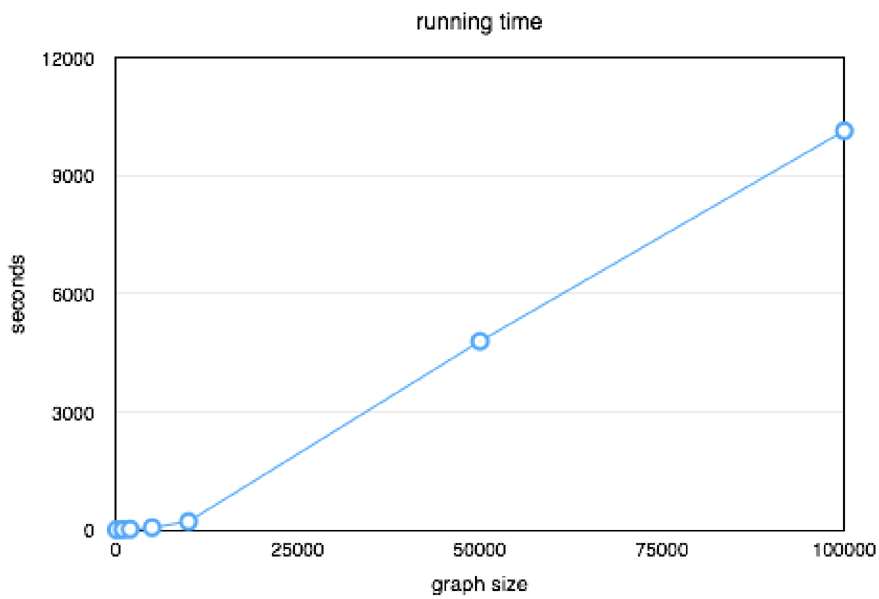


Figure 39: Running time VS graph size

4.10 Innovative task: Minimum Spanning Tree

4.10.1 Proof of Correctness

In order to verify the correctness of our implementation, we first run our algorithm on tiny synthetic graph with 10 nodes and 17 edges. We verified the result is correct. Then we compare our implementation against MATLAB's implementation of Minimum Spanning Tree and we get identical result for output of both implementation.

4.10.2 Experiment on Large datasets

Since MST algorithm require that graph is fully connected, weighted, and undirected. We can barely find such graph in Konect and SNAP project. Therefore, we generate synthetic graph of different size. We plot the runtime against graph size (number of nodes) in Figure 40. We find the run time grows near-quadratically as the number of nodes in graph. This is identical to the time complexity of Prim's Algorithm, which is $O(N^2)$.

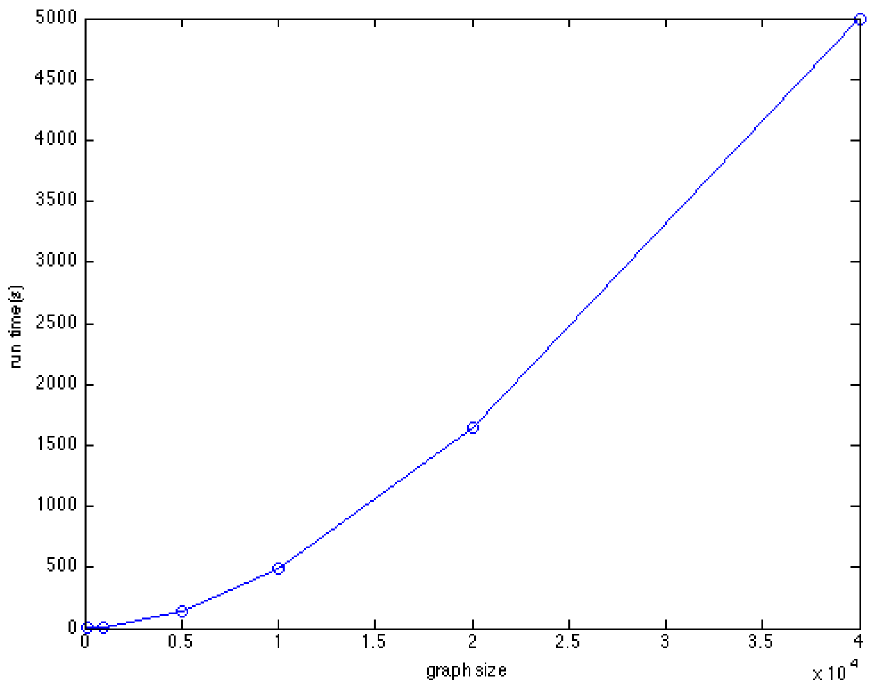


Figure 40: MST runtime plot

5 Conclusions

In this project, we investigate the major questions discussed in graph mining. We explored and solved the following problems:

- We summarize the importance of graph mining techniques and propose our approach to this problem, which is Relational Database Management System.
- We conduct extensive survey about 7 graph mining tasks, each team member has read at least six papers each for the tasks.
- For task 1, we calculate degree distribution of each node and do visualization. By observing the plot, we conclude that **social network** data exhibits **power law**, while this is not a general rule, for instance, Roadnet dataset doest not show power law.

- For task 2, we calculate the pagerank of each node in a graph, namely the importance of each node. By visualize the pagerank distribution in a rank-frequency plot, we again observe **power law**.

- For task 3, we compute the weakly connected components for a graph. By observing the plot, we find that there is a Giant Connected Component(GCC) which contains the majority of nodes in a graph. And the frequency-size plot exhibits **power law**.

- For task 4, we calculate the radius for every node in a graph using an approximate algorithm. By visualizing the result as radius plot, we find that most graphs has a **single-modal** or **bi-modal** radius distribution.

- For task 5. We tackle the problem of calculating eigenvalue of an adjacent matrix by Lanczos mathod. We build a Python matrix operation library which wraps low level linear algebra operations of SQL.

- For task 6. We implement the belief propagation using FABP algorithm. By conducting experiments on large datasets, we successfully perform **semi-supervised learning** using partially available labels to inference the labels of other nodes in the graph.

- For task 7. We deal with the problem of count triangle(global or local) in a graph by calculating the eigenvalue of the adjacent matrix. Through the rank-frequency plot of local triangle distribution, we again observe **power law**.

- We finished 2 innovative tasks, namely shortest path, minimum spanning tree.

- We provide proof of correctness for each task we implemented.

References

[1] P.F. Felzenszwalb and D.P. Huttenlocher. Efficient belief propagation for early vision. *International journal of computer vision*, 70(1):41–54, 2006.

[2] U. Kang, Duen Horng Chau, and Christos Faloutsos. Mining large graphs: Algorithms, inference, and discoveries. In *ICDE*, pages 243–254, 2011.

[3] U Kang, Duen Horng, et al. Inference of beliefs on billion-scale graphs. 2010.

[4] U. Kang, Mary McGlohon, Leman Akoglu, and Christos Faloutsos. Patterns on the connected components of terabyte-scale graphs. In *ICDM*, pages 875–880, 2010.

[5] U Kang, Brendan Meeder, and Christos Faloutsos. Spectral analysis for billion-scale graphs: Discoveries and implementation. In *Advances in Knowledge Discovery and Data Mining*, pages 13–25. Springer, 2011.

[6] U Kang, Charalampos Tsourakakis, and Christos Faloutsos. PEGASUS: A peta-scale graph mining system - implementation and observations. *ICDM*, December 2009.

[7] U. Kang, Charalampos E. Tsourakakis, Ana Paula Appel, Christos Faloutsos, and Jure Leskovec. Hadi: Mining radii of large graphs. *TKDD*, 5(2):8, 2011.

[8] Danai Koutra, Tai-You Ke, U. Kang, Duen Horng Chau, Hsing-Kuo Kenneth Pao, and Christos Faloutsos. Unifying guilt-by-association approaches: Theorems and fast algorithms. In *ECML/PKDD (2)*, pages 245–260, 2011.

[9] Cornelius Lanczos. *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators*. United States Governm. Press Office, 1950.

[10] Amy N Langville and Carl D Meyer. Deeper inside pagerank. *Internet Mathematics*, 1(3):335–380, 2004.

[11] Robert J. McEliece, David J. C. MacKay, and Jung-Fu Cheng. Turbo decoding as an instance of pearl's belief propagation algorithm. *Selected Areas in Communications, IEEE Journal on*, 16(2):140–152, 1998.

[12] Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. Pig latin: a not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1099–1110. ACM, 2008.

[13] Andrew Pavlo, Erik Paulson, Alexander Rasin, Daniel J Abadi, David J DeWitt, Samuel Madden, and Michael Stonebraker. A comparison of approaches to large-scale data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 165–178. ACM, 2009.

[14] Charalampos E Tsourakakis. Fast counting of triangles in large real networks without counting: Algorithms and laws. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 608–617. IEEE, 2008.

[15] J.S. Yedidia, W.T. Freeman, and Y. Weiss. Understanding belief propagation and its generalizations. *Exploring artificial intelligence in the new millennium*, 8:236–239, 2003.

A Appendix

A.1 Labor Division

The team performed the following tasks

- Implementation of Task 1,2 [Wei Chen]
- Implementation of Task 3 [Siping Ji]
- Implementation of Task 4,6. [Siping Ji]
- Implementation of Task 5,7. [Wei Chen]
- Implementation of Shortest Path(Innovative). [Wei Chen]
- Implementation of Minimum Spanning Tree. [Siping Ji]
- Set up testing framework in travis[4]. [Wei Chen]
- Data collection [Wei Chen, Siping Ji]
- Experiments on the real data [Wei Chen, Siping Ji]

A.2 Project Development

All the project related activity(code, report) are managed by Github[5], a collaborative development community. All tasks are mapped into *issues*, each phase is a *milestone*. We fork from a central repository, when we finish our assigned tasks(issue), we send a pull request to the central repository. We are responsible for code review each other's code, then merge into the central repository. It's easy to see each team member's contribution by review the history of pull request and commit. The project address is here.[6]

A.3 Plan

Phase 1

Task	Due	Member
Task1	10/07/13	Wei
Task2	10/07/13	Wei
Task3	10/07/13	Siping

Phase 2

Task4	11/05/13	Siping
Task5	11/05/13	Wei
Task6	11/05/13	Siping
Task7	11/10/13	Wei
Task8	11/10/13	Siping
Final	11/20/13	Wei, Siping
Poster	11/20/13	Wei, Siping

[4] travis-ci.org
[5] https://www.github.com
[6] https://github.com/essex405/graph-mining-rdbms

Phase 3		
Packaging code	11/30/13	Wei Chen, Siping
Final report	11/30/13	Wei Chen, Siping

A.4 Additional Task

In addition to the default tasks, we plan to complete another two: shortest path and minimum spanning tree. The detailed report please refer to **Experiments**.

A.5 Unit Tests

The host language we use is Python, thus we plan to use its internal unit test framework[7] as our testing module. We did unit tests for following modules:

- Matrix Vector multiplication, abstract out the some basic matrix-vector, matrix-matrix operations.
- SQL user defined function bit-or
- SQL user defined function fm-size

[7]http://docs.python.org/2/library/unittest.html

A.6 Source Code

A.6.1 ../src/bp/bp.py

```
import psycopg2
import sys

def time_it(fn):
        def wrapped(*args):
                import time
                start = time.time()
                fn(*args)
                used = time.time() - start
                print "%s used %s" % (str(fn), used)
        return wrapped

@time_it
def matrix_vector_multiply(conn, m1, m2, result):
        cur = conn.cursor()
        cur.execute("delete from %s" % result)
        cur.execute("insert into %s select A.row, sum(A.val * B.val) from %s
        conn.commit()
        cur.close()


def create_matrix(conn, table_name):
        cur = conn.cursor()
        drop_if_exists(conn, table_name)
        cur.execute("create table %s (row int, col int, val float)" % table_na
        conn.commit()
        cur.close()


def drop_if_exists(conn, table_name):
        cur = conn.cursor()
        cur.execute("drop table if exists %s" % table_name)
        conn.commit()
        cur.close()

def out_degree(conn, edge_table, degree_table, weighted):
        cur = conn.cursor()
        drop_if_exists(conn, degree_table)
        cur.execute("create table %s(id int, degree int)" % degree_table)
```

```
        if weighted == False:
                cur.execute("insert into %s select src_id, count(*) from %s g
        else:
                cur.execute("insert into %s select src_id, sum(weight) from %
        conn.commit()
        cur.close()

@time_it
def vector_add(conn, m1, m2):
        '''
        result is updated to m1
        '''
        cur = conn.cursor()
        #cur.execute('update %s set val = A.val + B.val from %s as A, %s as B
        cur.execute('delete from tmp');
        cur.execute('insert into tmp select A.id, A.val + B.val from %s as A,
        cur.execute('delete from %s' % m1)
        cur.execute('insert into %s select * from tmp' % m1)
        conn.commit()
        cur.close()

def create_vector(conn, tbl_name):
        cur = conn.cursor()
        drop_if_exists(conn, tbl_name)
        cur.execute('create table %s(id int, val float)' % tbl_name)
        conn.commit()

@time_it
def rand_init_matrix(conn, matrix, edge_table):
        cur = conn.cursor()
        cur.execute('insert into %s select src_id, rnd_prior() from %s group
        conn.commit()

def is_stablized(conn, belief, belief_new, threshold = 0.000001):
        cur = conn.cursor()
        cur.execute('select sqrt(sum(power(B.val - A.val, 2))) from %s as A,
        diff = cur.fetchone()[0]
        if diff < threshold:
                return True
        else:
                return False
```

```
def create_rnd_init(conn):
        fcn_def = \
                """
                CREATE or REPLACE function rnd_prior()
                returns float
                AS
                $$
                DECLARE
                retval float := 0;
                rnd_val float;
                BEGIN
                        rnd_val = random();
                        if rnd_val > 0.95 then
                                retval = 0.001;
                        elsif rnd_val < 0.05 then
                                retval = -0.001;
                        end if;
                        return retval;
                END;
                $$
                language plpgsql
                """
        cur = conn.cursor()
        cur.execute(fcn_def)
        conn.commit()

def summarize(conn, final_belief):
        cur = conn.cursor()
        cur.execute("select count(*) from %s where val > 0" % final_belief)
        pos = cur.fetchone()[0]

        cur.execute("select count(*) from %s where val = 0" % final_belief)
        zero = cur.fetchone()[0]

        cur.execute("select count(*) from %s where val < 0" % final_belief)
        neg = cur.fetchone()[0]
        print "postive: %d" % pos
        print "zero: %d" % zero
        print "negative: %d" % neg
```

```
def compute_bp(conn, edge_table, target_table, weighted = False):
        create_rnd_init(conn);
        cur = conn.cursor()
        h = 0.002
        a = 4.0 * (h ** 2) / (1 - 4 * h ** 2)
        c = 2.0 * h / (1 - 4 * h ** 2)
        W = "W"
        #W_new = "W_new"
        prior = "prior"
        belief = "bp_" + target_table
        belief_new = "belief_new"
        degree_table = "degree_table"
        out_degree(conn, edge_table, degree_table, weighted)
        create_matrix(conn, W)
        #create_matrix(conn, W_new)
        create_vector(conn, belief)
        create_vector(conn, belief_new)
        create_vector(conn, prior)
        create_vector(conn, 'tmp')
        rand_init_matrix(conn, prior, edge_table)
        if weighted == True:
                cur.execute("insert into %s select src_id, dst_id, weight * %
        else:
                cur.execute("insert into %s select src_id, dst_id, %f from %s
        conn.commit()

        cur.execute("insert into %s select id, id, -%f * degree from %s" % (W
        cur.execute("drop index if exists  w_index")
        cur.execute("create index w_index on %s(col)" % W)
        cur.execute("drop index if exists prior_index");
        cur.execute("create index prior_index on %s(id)" % prior)
        conn.commit()


        print "initialized"
        matrix_vector_multiply(conn, W, prior, belief)

        max_iteration = 10
        for i in range(max_iteration):
                print "iteration %d" % i
                #cur.execute("delete from %s" % W_new)
                matrix_vector_multiply(conn, W, belief, belief_new)
```

```
                vector_add(conn, belief_new, prior)
                if(is_stablized(conn, belief, belief_new)):
                        break
                cur.execute("delete from %s" % belief)
                cur.execute("insert into %s select * from %s" % (belief, beli
        conn.commit()

        summarize(conn, belief)


if __name__ == "__main__":
    conn = psycopg2.connect(database="mydb", host="127.0.0.1")
    compute_bp(conn, sys.argv[1], sys.argv[2])
    conn.close()
```

A.6.2 ../src/cc/assign.sql

```
create or replace function assign() returns void as
$$
BEGIN
update component_tmp
        set cid = ccid
        from (
                select component.nid as nnid, component.cid as ccid
                from component
                ) as cmpt
        where nid = nnid;
END;
$$
language plpgsql;
```

A.6.3 ../src/cc/cc.py

```
import psycopg2
import sys

def drop_if_exists(conn, table_name):
        cur = conn.cursor()
        cur.execute("drop table if exists %s" % table_name)
        conn.commit()
        cur.close()

def cc_init(conn, edge_table, target_table):
        cur = conn.cursor()
```

```
        drop_if_exists(conn, "component")
        drop_if_exists(conn, "component_tmp")
        drop_if_exists(conn, target_table)
        # cur.execute("drop index if exists c_index")
        # cur.execute("drop index if exists ct_index")
        cur.execute("drop index if exists edge_src_index")
        cur.execute("drop index if exists edge_dst_index")
        cur.execute("create table component (nid int not null unique, cid int
        cur.execute("create table component_tmp (nid int, cid int)")
        cur.execute("create table %s (nid int, cid int)" % target_table)
        cur.execute("insert into component_tmp(nid, cid) select distinct src_
        cur.execute("insert into component_tmp(nid, cid) select distinct dst_
        cur.execute("""
                insert into component (nid, cid) select distinct nid, cid fro
                drop table component_tmp;
                create table component_tmp (nid int, cid int);
                insert into component_tmp (nid, cid) select nid, cid from com

                create index edge_src_index on %s (src_id);
                create index edge_dst_index on %s (dst_id);
                """ % (edge_table, edge_table))
        conn.commit()
        print "initialized"
        cur.close()

def save_result(conn, target_table):
        cur = conn.cursor()
        cur.execute("insert into %s(nid, cid) select nid, cid from component
        conn.commit()
        cur.close()

def update(conn, edge_table):
        cur = conn.cursor()
        cur.execute("""
        update component
                set cid = new_cid
                from (
                        select dst_id as id, min(component.cid) as new_cid
                        from %s, component
                        where src_id = component.nid
                        group by id
                        ) as newComponent
```

```
                where nid = id and new_cid < cid;
        """ % edge_table)
        conn.commit()
        cur.execute("""
        update component
                set cid = new_cid
                from (
                        select src_id as id, min(component.cid) as new_cid
                        from %s, component
                        where dst_id = component.nid
                        group by id
                        ) as newComponent
                where nid = id and new_cid < cid;
        """ % edge_table)
        conn.commit()
        cur.close()

def count_diff(conn):
        cur = conn.cursor()
        cur.execute("""
                select count(*)
                from component, component_tmp
                where component.nid = component_tmp.nid and component.cid <>
        """)
        diff = cur.fetchone()[0]
        return diff

def cc_assign(conn):
        cur = conn.cursor()
        cur.execute("""
                update component_tmp
                set cid = ccid
                from (
                        select component.nid as nnid, component.cid as ccid
                        from component
                        ) as cmpt
                where nid = nnid;
                """)
        conn.commit()

def summarize(conn, target_table):
        cur = conn.cursor()
```

```
        cur.execute("select count(distinct cid) from %s" % target_table)
        num_cc = cur.fetchone()[0]
        cur.execute("select max(cnt) from (select count(*) as cnt from %s gro
        max_cc = cur.fetchone()[0]
        print "number of connected components:%d" % num_cc
        print "largest connected components:%d vertices" % max_cc
        drop_if_exists(conn, "tmp")
        cur.execute("select cnt, count(*) into tmp from (select count(*) as c
        print "size\tcount"
        f = open(target_table + '.csv', 'w')
        cur.copy_to(sys.stdout, 'tmp', sep = "\t")
        cur.copy_to(f, 'tmp', sep = ",")
        cur.close()

def compute_cc(conn, edge_table, target_table):
        cc_init(conn, edge_table, target_table)
        update(conn, edge_table)
        diff = count_diff(conn)
        iter = 1
        while diff > 0:
                print "iteration %d %d" % (iter, diff)
                cc_assign(conn)
                update(conn, edge_table)
                diff = count_diff(conn)
                iter = iter + 1
        save_result(conn, target_table)
        summarize(conn, target_table)



if __name__ == "__main__":
        conn = psycopg2.connect(database="mydb", host="127.0.0.1")
        compute_cc(conn, sys.argv[1], "cc_" + sys.argv[2])
        conn.close()
```

A.6.4 ../src/cc/count_diff.sql

```
create or replace function count_diff() returns integer AS
$$
DECLARE
        diff_count integer := 0;
BEGIN
        select into diff_count
                count(*)
```

```
                from component, component_tmp
                where component.nid = component_tmp.nid and component.cid <>

        return diff_count;
END;
$$
language plpgsql;
```

A.6.5 ../src/cc/init.sql

```
create or replace function init_component() returns void AS
$$
BEGIN
drop table if exists component;
drop table if exists component_tmp;
drop index if exists c_index;
drop index if exists ct_index;
create table component (nid int not null unique, cid int);
create table component_tmp (nid int, cid int);
insert into component_tmp(nid, cid) select distinct src_id, src_id from edge;
insert into component_tmp(nid, cid)     select distinct dst_id, dst_id from e
insert into component (nid, cid) select distinct nid, cid from component_tmp;
drop table component_tmp;
create table component_tmp (nid int, cid int);
insert into component_tmp (nid, cid) select nid, cid from component;
create index c_index on component (nid);
create index ct_index on component_tmp (nid);
RAISE NOTICE 'component table initialization complete.';
END;
$$
language plpgsql;
```

A.6.6 ../src/cc/main.sql

```
create or replace function cc_main() returns void AS
$$
DECLARE
iter integer := 1;
diff_count integer := 0;
BEGIN
        perform init_component();
        perform update_component();
        select into diff_count count_diff();
        while diff_count > 0 loop
```

```
                RAISE NOTICE 'iteration: %, diff_count: %', iter, diff_count;
                perform assign();
                perform update_component();
                select into diff_count count_diff();
                iter := iter + 1;
        end loop;
        perform cc_stat();
END;
$$
language plpgsql;
```

A.6.7 ../src/cc/stat.sql

```
create or replace function cc_stat() returns void as
$$
DECLARE
num_cc integer := 0;
max_node integer := 0;
BEGIN
select into num_cc count(distinct cid) from component;
RAISE NOTICE 'number of connected components: %', num_cc;
select into max_node count(*) from component group by cid order by count(*) d
RAISE NOTICE 'number of nodes in max WSC: %', max_node;
END;
$$
language plpgsql;
```

A.6.8 ../src/cc/update.sql

```
-- component id is the minimum node id of all nodes in the component
-- component table(nid, cid)
-- since we are computing weakly connecte components
CREATE or REPLACE FUNCTION update_component() returns void AS
$$
BEGIN
update component
        set cid = new_cid
        from (
                select dst_id as id, min(component.cid) as new_cid
                from edge, component
                where src_id = component.nid
                group by id
                ) as newComponent
        where nid = id and new_cid < cid;
```

```
update component
        set cid = new_cid
        from (
                select src_id as id, min(component.cid) as new_cid
                from edge, component
                where dst_id = component.nid
                group by id
                ) as newComponent
        where nid = id and new_cid < cid;
END;
$$
language plpgsql;
```

A.6.9 ../src/common/basic_operation.py

```
##############################################
# creation & destroy
##############################################
def drop_if_exists(tbl_name, conn):
    """
    delete a table if exists
    """
    cur = conn.cursor()
    cur.execute("drop table if exists %s" % tbl_name)
    conn.commit()

def create_vector_or_matrix(tbl_name, conn):
    """
    create an *empty* vector or matrix
    """
    drop_if_exists(tbl_name, conn)
    cur = conn.cursor()
    cur.execute("create table %s (row int, col int, value float)" % tbl_name)
    conn.commit()

def initilizat_vector(tbl_name, dim, conn):
    """
    initilize a vector
    """
    import random
    clear_table(tbl_name, conn)
    cur = conn.cursor()
```

```
    # this is slow, I know
    for i in range(dim):
        cur.execute("insert into %s values (%s, 0, %s)" % (tbl_name, i, 0.0))
    conn.commit()

def initilizat_square_matrix(tbl_name, dim, conn):
    """
    initilize a matrix
    """
    import random
    clear_table(tbl_name, conn)
    cur = conn.cursor()
    # this is slow, I know
    for i in range(dim):
        for j in range(dim):
            cur.execute("insert into %s values (%s, %s, %s)" % (tbl_name, i,
    conn.commit()

def random_vector(tbl_name, dim, conn):
    """
    randomly initilize a vector
    """
    import random
    clear_table(tbl_name, conn)
    cur = conn.cursor()
    # this is slow, I know
    for i in range(dim):
        cur.execute("insert into %s values (%s, 0, %s)" % (tbl_name, i, rando
    conn.commit()

def random_square_matrix(tbl_name, dim, conn):
    """
    randomly initilize a matrix
    """
    import random
    clear_table(tbl_name, conn)
    cur = conn.cursor()
    # this is slow, I know
    for i in range(dim):
        for j in range(dim):
            cur.execute("insert into %s values (%s, %s, %s)" % (tbl_name, i,
    conn.commit()
```

```
def assign_to(from_tbl, to_tbl, conn):
    """
    assign a vector/matrix to another variable(table)
    """
    clear_table(to_tbl, conn)
    cur = conn.cursor()
    cur.execute("insert into %s select * from %s" % (to_tbl, from_tbl))
    conn.commit()


##############################################
# read
##############################################
def vector_length(tbl_name, conn):
    """
    normalization length
    """
    cur = conn.cursor()
    cur.execute("select sqrt(sum(power(value, 2))) from %s" % tbl_name)
    return cur.fetchone()[0]

def rank_length(tbl_name, conn):
    """
    normalization length
    """
    cur = conn.cursor()
    cur.execute("select sqrt(sum(power(rank, 2))) from %s" % tbl_name)
    return cur.fetchone()[0]

def vector_dot_product(a, b, conn):
    """
    a .X b
    """
    cur = conn.cursor()
    cur.execute("select sum(A.value * B.value) from %s A, %s B where A.row =
    return cur.fetchone()[0]

##############################################
# write
##############################################
def reverse_edge(tbl, conn):
```

```
    cur = conn.cursor()
    cur.execute("insert into %s select to_id, from_id, value from %s" % (tbl,
    conn.commit()

def reverse_matrix(tbl, conn):
    cur = conn.cursor()
    cur.execute("insert into %s select col, row, value from %s" % (tbl, tbl))
    conn.commit()

def clear_table(tbl_name, conn):
    cur = conn.cursor()
    cur.execute("delete from %s" % tbl_name)
    conn.commit()

def matrix_multiply_matrix_overwrite(a, b, result, conn):
    create_vector_or_matrix(result, conn)
    cur = conn.cursor()
    cur.execute("insert into %s select A.row, B.col, sum(A.value * B.value) f
    conn.commit()

def matrix_multiply_vector_overwrite(a, b, c, conn):
    """
    a X b = c
    """
    clear_table(c, conn)
    cur = conn.cursor()
    cur.mogrify("insert into %s select A.row, 0, sum(A.value * B.value) from
    cur.execute("insert into %s select A.row, 0, sum(A.value * B.value) from
    conn.commit()

def matrix_multiply_matrix_ignore(a, b, result, conn):
    """
    seem like not easy to do with PostSQL
    """
    pass


def normalize_vector(tbl_name, conn):
    """
    normalized vector
    """
    l = vector_length(tbl_name, conn)
```

```
    cur = conn.cursor()
    cur.execute("update %s set value = value / %s" % (tbl_name, l))
    conn.commit()

def normalize_pagerank(tbl_name, conn):
    """
    normalized pagerank
    """
    l = rank_length(tbl_name, conn)
    cur = conn.cursor()
    cur.execute("update %s set rank = rank / %s" % (tbl_name, l))
    conn.commit()

def normalize_column(tbl_name, col, conn):
    """
    normalize a column of a matrix
    """
    cur = conn.cursor()
    cur.execute("select sqrt(sum(power(value, 2))) from %s where col = %s" %
    l = cur.fetchone()[0]
    if l > 0:
        cur.execute("update %s Q1 set value = value / %s where col = %s" % (t
    conn.commit()

def devide_column(tbl_name, col, d, conn):
    """
    devide a column of a matrix by a constance
    """
    cur = conn.cursor()
    cur.execute("update %s Q1 set value = value / %s where col = %s" % (tbl_na
    conn.commit()

def set_matrix(tbl_name, row, col, v, conn):
    """
    update a cell in matrix
    """
    cur = conn.cursor()
    cur.execute("update %s set value = %s where row = %s and col = %s" % (tbl_
    conn.commit()
```

A.6.10 ../src/common/util.py

```
########################
```

```
# Load file
########################e
def load_undirected_graph_into_table(tbl_name, filename, doreverse, conn):
    """ load an undirected unweighted file into a (from_id, to_id) pair"""
    cur = conn.cursor()
    cur.execute("drop table if exists %s" % tbl_name)
    cur.execute("create table %s(from_id int, to_id int)" % tbl_name)
    conn.commit()
    with open(filename) as f:
        cur.copy_from(f, tbl_name, columns=('from_id', 'to_id'))
    if doreverse:
        cur.execute("insert into %s select to_id, from_id from %s" % (tbl_nam
    conn.commit()


def load_unweighted_graph(tbl_name, filename, doreverse, conn, separator = "\
    """ load an unweighted file into a (src_id, dst_id) pair"""
    cur = conn.cursor()
    cur.execute("drop table if exists %s" % tbl_name)
    cur.execute("create table %s(src_id int, dst_id int)" % tbl_name)
    conn.commit()
    with open(filename) as f:
        cur.copy_from(f, tbl_name, columns=('src_id', 'dst_id'), sep = separa
    if doreverse:
        cur.execute("insert into %s ((select dst_id, src_id from %s) except (
    conn.commit()

def load_weighted_graph(tbl_name, filename, doreverse, conn, separator = "\t"
    """ load an weighted file into a (src_id, dst_id, weight) pair"""
    cur = conn.cursor()
    cur.execute("drop table if exists %s" % tbl_name)
    cur.execute("create table %s(src_id int, dst_id int, weight float)" % tbl_
    conn.commit()
    with open(filename) as f:
        cur.copy_from(f, tbl_name, columns=('src_id', 'dst_id', 'weight'), se
    if doreverse:
        cur.execute("insert into %s ((select dst_id, src_id, weight from %s)
    conn.commit()

def convert_matrix_from_graph():
    pass
```

A.6.11 ../src/ddis/ddis.py

```
def indis(tbl_name, result, conn):
    """ in degree distribution"""
    cur = conn.cursor()
    cur.execute("drop table if exists %s" % result)
    cur.execute("create table %s(deg int, cnt int)" % result)
    cur.execute("INSERT into %s SELECT degree, count(*) \
                FROM ( SELECT count(*) as degree FROM %s GROUP BY to_id ) as
                GROUP BY degree order by degree ASC" % (result, tbl_name))
    conn.commit()

def outdis(tbl_name, result, conn):
    """ out degree distribution """
    cur = conn.cursor()
    cur.execute("drop table if exists %s" % result)
    cur.execute("create table %s(deg int, cnt int)" % result)
    cur.execute("INSERT into %s SELECT degree, count(*) \
                FROM ( SELECT count(*) as degree FROM %s GROUP BY from_id )
                GROUP BY degree order by degree ASC" % (result, tbl_name))
    conn.commit()

def undirect_dis(tbl_name, result, conn):
    """ degree distribution """
    cur = conn.cursor()
    cur.execute("drop table if exists %s" % result)
    cur.execute("create table %s(deg int, cnt int)" % result)
    cur.execute("INSERT into %s SELECT degree, count(*) \
                FROM ( SELECT count(*) as degree FROM %s GROUP BY from_id )
                GROUP BY degree order by degree ASC" % (result, tbl_name))
    conn.commit()
```

A.6.12 ../src/eigenvalue/eigen_quodratic.py

```
from common.basic_operation import *
from qr_decompose import *

def eigen_quodratic(tbl_name, q, r, dim, conn):
    """
    apply QR factorization until some maximum iteration reached
    """
    create_vector_or_matrix(q, conn)
    create_vector_or_matrix(r, conn)
    for k in range(21):
```

```
        print "QR decomposition: %s" % k
        qr_decompose(tbl_name, q, r, dim, conn)
        matrix_multiply_matrix_overwrite(r, q, tbl_name, conn)
    matrix_multiply_matrix_overwrite(q, r, tbl_name, conn)
    assign_to(tbl_name, r, conn)
```

A.6.13 ../src/eigenvalue/lanczos.py

```
from common.basic_operation import *
from eigen_quodratic import *

def lanczos(A, b, n, m, conn):
    """
    A: n X n matrix
    b: initial vector
    m: number of steps
    """
    beta = "beta"
    v = ["v%s" % i for i in range(m+2)] # TODO: why this fixed?
    v_tmp = "v_tmp"
    alpha = "alpha"

    create_vector_or_matrix(beta, conn) # just empty
    for v_table in v:
        create_vector_or_matrix(v_table, conn) # just empty
    create_vector_or_matrix(v_tmp, conn) # just empty
    create_vector_or_matrix(alpha, conn) # just empty
    initilizat_vector(beta, m+1, conn) # beta_0 = 0
    initilizat_vector(v[0], n, conn) # v_0 = {0}
    initilizat_vector(alpha, m+1, conn)
    create_vector_or_matrix(v[1], conn) # empty v1
    assign_to(b, v[1], conn) # v_1 = b
    normalize_vector(v[1], conn) # v_1 = b/|b|
    index_counter = 0
    for i in range(1, m):
        print "Iteration: %s" % i
        matrix_multiply_vector_overwrite(A, v[i], v_tmp, conn) # v = A * v_i
        alpha_i = vector_dot_product(v[i], v_tmp, conn) # alpha_i = v_i * v
        set_matrix(alpha, i, 0, alpha_i, conn)

        cur = conn.cursor()
        cur.execute("select value from %s where row = %s" % (beta, i-1))
        beta1 = cur.fetchone()[0]
```

```
        cur.execute("select value from %s where row = %s" % (alpha, i))
        alpha1 = cur.fetchone()[0]
        cur.execute("create index vindex%s ON %s (row)" % (index_counter, v[i
        index_counter += 1
        cur.execute("create index vindex%s ON %s (row)" % (index_counter, v[i
        index_counter += 1
        print "index builded...."
        cur.execute("""
            update %s set value =
            value - %s * (select value from %s where row = %s.row)
                  - %s * (select value from %s where row = %s.row)""" % \
                  (v_tmp, beta1, v[i-1], v_tmp, alpha1, v[i], v_tmp))
        print "vector update done"
        vl = vector_length(v_tmp, conn) # |v|
        set_matrix(beta, i, 0, vl, conn) # beta_i = |v|
        if (vl == 0):
            break
        create_vector_or_matrix(v[i+1], conn) # just empty
        assign_to(v_tmp, v[i+1], conn) # v_i+1 = v
        cur = conn.cursor()
        cur.execute("update %s set value = value / (select value from %s wher
    cur = conn.cursor()
    V_mat = "v_matrix"
    create_vector_or_matrix(V_mat, conn)
    print "The width is %s" % len(v)
    for i in range(m):
        vvv = v[i]
        print "appending %s" % vvv
        cur.execute("insert into %s select row, %s, value from %s" % (V_mat,
        drop_if_exists(vvv, conn)
    ritz_vector(alpha, beta, m, V_mat, conn)
    drop_if_exists('alpha', conn)
    drop_if_exists('beta', conn)
    drop_if_exists('b', conn)
    drop_if_exists('t', conn)
    drop_if_exists('v', conn)
    drop_if_exists('v_tmp', conn)

def ritz_vector(alpha, beta, m, V, conn):
    """
    solve eigenvector for a smaller martix
    """
```

```
    t = "t"
    print "Building triagonal matrix"
    build_tridiagonal_matrix(alpha, beta, m, t, conn)
    print "QR decomposition"
    eigen_quodratic(t, 'eigenvec_tmp', 'eigenval', m, conn)
    print "Calculating eigen vectors....."
    matrix_multiply_matrix_overwrite(V, 'eigenvec_tmp', 'eigenvec', conn)
    drop_if_exists("eigenvec_tmp", conn)
    print "Eigenvector calculated, they are stored in %s and %s" % ('eigenval

def build_tridiagonal_matrix(alpha, beta, m, t, conn):
    create_vector_or_matrix(t, conn) # just empty
    cur = conn.cursor()
    for i in range(1, m+1):
        for j in range(1, m+1):
            if j == i:
                # print cur.mogrify("insert into %s select %s, %s, value from
                cur.execute("insert into %s select %s, %s, value from %s wher
            elif j == i + 1:
                cur.execute("insert into %s select %s, %s, value from %s wher
            elif i - 1 >= 1 and j == i - 1:
                cur.execute("insert into %s select %s, %s, value from %s wher
            else:
                # print cur.mogrify("insert into %s values (%s, %s, 0.0)" % (
                cur.execute("insert into %s values (%s, %s, 0.0)" % (t, i-1,
    conn.commit()
```

A.6.14 ../src/eigenvalue/qr_decompose.py

```
from common.basic_operation import *

## QR factorization of matrix
def qr_decompose(v, q, r, dim, conn):
    """
    v: original matrix
    q: left matrix
    r: right matrix
    dim: dimension of v

    1. assume q,r are empty
    2. v is square matrix
    """
    # copy v to q
```

```
    cur = conn.cursor()
    cur.execute("delete from %s" % q)
    cur.execute("delete from %s" % r)
    cur.execute("insert into %s select * from %s" % (q, v))
    conn.commit()
    for i in range(0, dim):
        cur.execute("insert into %s select %s, %s, sqrt(sum(power(value, 2)))
        conn.commit()
        normalize_column(q, i, conn)

        for j in range(i+1, dim):
            cur.execute("(select sum(Q1.value * Q2.value) from %s Q1, %s Q2 w
            r_i_j = cur.fetchone()[0]
            cur.execute("insert into %s values (%s, %s, %s)" % (r, i, j, r_i_
            cur.execute("update %s Q1 set value = value - %s * (select value
            conn.commit()
```

A.6.15 ../src/eigenvalue/test_data.txt

```
(0.368701362076, 0.676493825912, 0.407250855446), (0.501224330792, 0.44873746



(0, 0, 0.368701362076), (0, 1, 0.676493825912), (0, 2, 0.407250855446), (1, 0



[[0.5, 0.2, 0.3];[0.1, 0.7, 0.3];[0.1, 0.2, 0.5]]

[[0.5, 0.2, 0.3];[0.1, 0.7, 0.3];[0.1, 0.2, 0.5]]
```

A.6.16 ../src/generate_appendix.sh

```
find . -type f ! -regex ".*/\..*" ! -name ".*" ! -name "*~" ! -name 'src2pdf'
sed 's/^\..//' |                ## Change ./foo/bar.src to foo/bar.src

while read  i; do               ## Loop through each file

  ## This command will include the file in the PDF
   echo "\subsubsection{../src/$i}"
   echo "\lstinputlisting{../src/$i}"
done
```

A.6.17 ../src/graph_generator.py

```
import random

from common.basic_operation import *

def generate_directed_graph(name, dim, conn):
    """ generate a pseudot random graph """
    cur = conn.cursor()
    drop_if_exists(name, conn)
    cur.execute("create table %s (from_id int, to_id int, value real)" % name
    for i in range(5, dim):
        cands = set(random.sample(xrange(i), 5))
        for f in cands:
            cur.execute("insert into %s values (%s, %s, %s)" % (name, f, i, r

def generate_undirected_graph(name, dim, conn):
    """ generate a pseudot undirected random graph """
    cur = conn.cursor()
    drop_if_exists(name, conn)
    cur.execute("create table %s (src_id int, dst_id int, weight float)" % na
    for i in range(5, dim):
        cands = set(random.sample(xrange(i), 5))
        for f in cands:
            cur.execute("insert into %s values (%s, %s, %s)" % (name, f, i, r
    cur.execute("insert into %s ((select dst_id, src_id, weight from %s) exce
    conn.commit()
```

A.6.18 ../src/misc/pagerank.py

```
import numpy as np
from scipy.sparse import csc_matrix
import math


def pageRank(G, s = .85, maxerr = .001):
    """
    Computes the pagerank for each of the n states.

    Used in webpage ranking and text summarization using unweighted
    or weighted transitions respectively.


    Args
    ----------
```

```
G: matrix representing state transitions
   Gij can be a boolean or non negative real number representing the
   transition weight from state i to j.

Kwargs
----------
s: probability of following a transition. 1-s probability of teleporting
   to another state. Defaults to 0.85

maxerr: if the sum of pageranks between iterations is bellow this we will
        have converged. Defaults to 0.001
"""
n = G.shape[0]

# transform G into markov matrix M
M = csc_matrix(G,dtype=np.float)
rsums = np.array(M.sum(1))[:,0]
ri, ci = M.nonzero()
M.data /= rsums[ri]

# bool array of sink states
sink = rsums==0

# Compute pagerank r until we converge
ro, r = np.zeros(n), np.ones(n)
while np.sum(np.abs(r-ro)) > maxerr:
    ro = r.copy()
    # calculate each pagerank at a time
    for i in xrange(0,n):
        # inlinks of state i
        Ii = np.array(M[:,i].todense())[:,0]
        # account for sink states
        Si = sink / float(n)
        # account for teleportation to state i
        Ti = np.ones(n) / float(n)

        r[i] = ro.dot( Ii*s + Si*s + Ti*(1-s) )

# return normalized pagerank
s = 0.0
for i in r:
    s += i * i
```

```
    return r/math.sqrt(s)



if __name__=='__main__':
    # Example extracted from 'Introduction to Information Retrieval'
    G = np.array([[0,0,1,0,0,0,0],
                  [0,1,1,0,0,0,0],
                  [1,0,1,1,0,0,0],
                  [0,0,0,1,1,0,0],
                  [0,0,0,0,0,0,1],
                  [0,0,0,0,0,1,1],
                  [0,0,0,1,1,0,1]])

    print pageRank(G,s=.85)
```

A.6.19 ../src/mst/mst.py

```
import psycopg2
import sys

def drop_if_exists(tbl_name, conn):
    """
    delete a table if exists
    """
    cur = conn.cursor()
    cur.execute("drop table if exists %s" % tbl_name)
    conn.commit()


def mst(conn, edge_table, dataset):
        """
        Prim's algorithm
        """
        cur = conn.cursor()
        target_table = "mst_" + dataset
        node_table = "node_mst"
        tmp_table = "tmp_table"
        drop_if_exists(target_table, conn)
        drop_if_exists(node_table, conn)
        drop_if_exists(tmp_table, conn)
        #cur.execute("drop index if exists e_index")
```

```
        #cur.execute("create index e_index on %s(src_id)" % edge_table)
        cur.execute("create table %s(src_id int, dst_id int, weight float)" %
        cur.execute("create table %s(nid int)" % node_table)
        cur.execute("create table %s(src_id int, dst_id int, weight float)" %
        conn.commit()
        cur.execute('select count(distinct src_id)from %s' % edge_table)
        num_nodes = cur.fetchone()[0]
        # randomly insert an initial node
        cur.execute('insert into %s select src_id from %s limit 1' % (node_ta
        conn.commit()
        for i in range(num_nodes - 1):
                print "iteration %d" % (i + 1)
                cur.execute("""insert into %s select src_id, dst_id, weight f
                        where A.src_id = B.nid AND A.dst_id not in (select ni
                cur.execute('insert into %s select dst_id from %s' % (node_ta
                cur.execute('insert into %s select * from %s' % (target_table
                cur.execute('delete from %s' % tmp_table)
                conn.commit()

if __name__ == "__main__":
        conn = psycopg2.connect(database="mydb", host="127.0.0.1")
        mst(conn, sys.argv[1], sys.argv[2])
```

A.6.20 ../src/mst/test.dat

```
1 2 6
2 1 6
1 4 5
4 1 5
1 3 1
3 1 1
2 3 5
3 2 5
2 5 3
5 2 3
3 4 5
4 3 5
3 5 6
5 3 6
3 6 4
6 3 4
4 6 2
6 4 2
```

```
5 6 6
6 5 6
```

A.6.21 ../src/pagerank/pagerank.py

```
import query

def calculatepagerank(graph, rank, conn):
    cur = conn.cursor()
    cur.execute("drop function if exists calc_pagerank()")
    cur.execute(query.rank_function % (rank, rank, graph, graph, rank, rank,
   # print cur.mogrify(query.rank_function % (rank, rank, graph, graph, rank
    print "ranking....."
    cur = conn.cursor();
    cur.execute("select calc_pagerank()")
    conn.commit()
```

A.6.22 ../src/pagerank/query.py

```
rank_function = '''
CREATE OR REPLACE FUNCTION calc_pagerank() RETURNS VOID AS
$$
DECLARE
    currentIndex integer;
    diff real;
    num_of_nodes integer;
    damper real;
    nrank real;
    nid integer;
BEGIN
    RAISE NOTICE 'Entering....';
    drop table if exists %s;
    drop table if exists pagerank_tmp;
    drop table if exists out_degree;
    drop table if exists trans;

    -- first insert all nodes into a tmp table allow duplicate, then clean up
    create table pagerank_tmp(node_id int, rank real);
    create table %s(node_id int primary key, rank real);
    insert into pagerank_tmp(node_id, rank) select from_id, 1.0 from %s;
    insert into pagerank_tmp(node_id, rank) select to_id, 1.0 from %s;
    select count(*) into num_of_nodes from pagerank_tmp group by node_id;
    insert into %s(node_id, rank) select node_id, 1.0/num_of_nodes from pager
    delete from pagerank_tmp;
```

```
    insert into pagerank_tmp(node_id, rank) select node_id,rank from %s;

    -- init out degree
    create table out_degree(node_id int primary key, degree int);
    insert into out_degree(node_id, degree) select from_id, count(*) from %s

    -- init transfer weight matrix
    create table trans(to_id int, from_id int, weight real);
    insert into trans select to_id, from_id, 1.0/OD.degree from %s, out_degre

    currentIndex := 0;
    damper := 0.85;
    create index to_index on trans(to_id);
    create index from_index on trans(from_id);
    create index prtmp_index on pagerank_tmp(node_id);
    while currentIndex < 10 loop
        RAISE NOTICE 'currentIndex: %s', currentIndex;

        for nid, nrank in
        select trans.to_id, (1 - damper)/num_of_nodes + damper * sum(trans.we
        from trans, %s where trans.from_id = %s.node_id
        group by trans.to_id loop
            update pagerank_tmp set rank = nrank where node_id = nid;
        end loop;

        select sum( (N.rank - O.rank) * (N.rank - O.rank) ) into diff
        from %s as O, pagerank_tmp as N where O.node_id = N.node_id;

        delete from %s;
        insert into %s select * from pagerank_tmp;

        currentIndex := currentIndex + 1;
        if diff < 0.0001 then
            exit;
        end if;
    end loop;
    drop table pagerank_tmp;
    drop table trans;
    drop table out_degree;
END;
$$
LANGUAGE plpgsql;
```

```
'''
```

A.6.23 ../src/radius/functions.py

```
import psycopg2

# user defined functions
def create_bit_or(conn):
        func_def = \
                '''
                create or replace function bit_or(int[], int[])
                returns int[]
                AS
                $$
                declare
                bs1 ALIAS for $1;
                bs2 ALIAS for $2;
                retval int[];
                BEGIN
                        for i in array_lower(bs1, 1)..array_upper(bs2, 1) LOC
                                retval[i] := bs1[i] | bs2[i];
                        end loop;
                        return retval;
                END;
                $$
                language plpgsql
                STRICT
                '''
        create_function(conn,func_def)

def create_agg_bit_or(conn):
        cur = conn.cursor()
        cur.execute("drop aggregate if exists agg_bit_or(int[])")
        func_def = \
        '''
        CREATE AGGREGATE agg_bit_or(int[])
        (
                sfunc = bit_or,
                stype = int[]
        );
        '''
        create_function(conn,func_def)
```

```
def create_fm_assign(conn):
        func_def = \
        """
        create or replace function fm_assign(int)
        returns BIT[]
        AS
        $$
        declare
        dim ALIAS for $1;
        retVal bit(32)[];
        bitStr bit(32);
        rndNum numeric;
        index int;
        BEGIN
                for i in 1..dim LOOP
                        retVal[i] := 0::bit(32);
                        select random() into rndNum;
                        select floor(log(0.5, rndNum)) into index;
                        select set_bit(retVal[i], index, 1) into bitStr;
                        retVal[i] := bitStr;
                        raise notice '% % %',bitStr, rndNum, index;
                end loop;
                return retVal;
        END;
        $$
        language plpgsql
        """
        create_function(conn ,func_def)

def create_fm_size(conn):
        func_def = \
        """
        create or replace function fm_size(bit[])
        returns numeric
        AS
        $$
        declare
        size numeric;
        phi numeric := 0.77351;
        bit_arr ALIAS for $1;
        r int := 0;
        k int := 0;
```

```
        BEGIN
                for i in array_lower(bit_arr, 1)..array_upper(bit_arr, 1) loo
                        r := r + msb(bit_arr[i]);
                        k = k + 1;
                end loop;
                raise notice '%', r;
                select power(2, 1.0 * r/k) / phi into size;
                return size;
        END;
        $$
        language plpgsql;
        """
        create_function(conn ,func_def)

def create_msb(conn):
        func_def = \
        """
        create or replace function msb(bit)
        returns int
        AS
        $$
        declare
        len int;
        bitStr alias for $1;
        tmpBit int;

        BEGIN
                select length(bitStr) into len;
                for i in 1..len LOOP
                        select get_bit(bitStr, i - 1) into tmpBit;
                        if tmpBit = 0 then
                                return i;
                        end if;
                end loop;
        END;
        $$
        language plpgsql
        """
        create_function(conn ,func_def)


def create_function(conn, func_def):
```

```
        cur = conn.cursor()
        cur.execute(func_def)
        cur.close()
        conn.commit()

def init_udf(conn):
        create_bit_or(conn)
        create_agg_bit_or(conn)
        create_msb(conn)
        create_fm_size(conn)
        create_fm_assign(conn)

if __name__ == '__main__':
        conn = psycopg2.connect(database="mydb", host="127.0.0.1")

        conn.close()
```

A.6.24 ../src/radius/hop.sql

```
--hop.sql
select hops.id, min(hops.hop) from hops, (select id, max(size) as max_size fro
where hops.id = foo.id and hops.size = foo.max_size
group by hops.id;
```

A.6.25 ../src/radius/radius.py

```
import sys
import random
import math

def time_it(fn):
        def wrapped(*args):
                import time
                start = time.time()
                fn(*args)
                used = time.time() - start
                print "%s used %s" % (str(fn), used)
        return wrapped

@time_it
def assign_fm(conn, tbl_name, edge_table, k):
        cur = conn.cursor()
        # cur.execute("delete from %s" % tbl_name)
        # cur.execute("insert into %s select src_id, fm_assign(%d) from %s gr
```

```
        cur.execute("select distinct src_id from %s"% edge_table)
        records = cur.fetchall()
        query = "insert into " + tbl_name + " values(%s, %s)"
        for i in range(len(records)):
                src_id = records[i][0]
                fm_string = generate_k_fm_strings(k)
                cur.execute(query, (src_id, fm_string))

        conn.commit()

        cur.close()

def generate_k_fm_strings(k):
        return [generate_fm_string() for i in range(k)]

def generate_fm_string():
        ran = random.random()
        index = math.floor(math.log(ran, 0.5))
        if index > 31:
                index = 31
        return 1<<int(index)



@time_it
def update_bitstring(conn, edge_table, vertex_table, tmp_table):
        cur = conn.cursor()
        #cur.execute("delete from %s" % tmp_table)
        cur.execute("update %s as A set fm = foo.fm from (select src_id as id
        conn.commit()
        cur.close()

def is_stablized(conn, vertex_table, tmp_table):
        cur = conn.cursor()
        diff = cur.execute("select count(*) from %s as A, %s as B where A.id
        diff = cur.fetchone()[0]
        print diff
        if diff == 0:
                return True
        else:
                return False
```

```
def drop_if_exists(conn, table_name):
        cur = conn.cursor()
        cur.execute("drop table if exists %s" % table_name)
        conn.commit()
        cur.close()



@time_it
def summarize(conn, radius_table, dataset):
        cur = conn.cursor()
        drop_if_exists(conn, "tmp")
        cur.execute("select radius, count(*) into tmp from %s group by radius
        f = open('radius_' + dataset + ".csv", 'w')
        cur.copy_to(sys.stdout, "tmp", sep = '\t')
        cur.copy_to(f, "tmp", sep = ',')

def compute_radius(conn, edge_table, dataset, s):
        """
        edge_table: edge table
        """
        print "computing radius.."
        vertex_table = "vertex"
        tmp_table = "tmp_v"
        hop_table = "hops"
        tmp_edge = "tmp_edge"
        radius_table = "radius_" + dataset
        cur = conn.cursor()
        drop_if_exists(conn, tmp_table)
        drop_if_exists(conn, vertex_table)
        drop_if_exists(conn, tmp_edge)
        drop_if_exists(conn, radius_table)
        drop_if_exists(conn, hop_table)
        cur.execute("create table %s(src_id int, dst_id int)" % tmp_edge)
        cur.execute("insert into %s select src_id, dst_id from %s" % (tmp_edg
        cur.execute("insert into %s select src_id, src_id from %s group by sr
        #cur.execute("insert into %s select dst_id, dst_id from %s where dst_
        #cur.execute("drop index if exists radius_index ")
        #cur.execute("create index radius_index on %s (src_id) " % edge_table

        cur.execute("create table %s(id int, fm int[])" % vertex_table)
        cur.execute("create table %s(id int, fm int[])" % tmp_table)
```

```
cur.execute("create table %s(id int, radius int)" % hop_table)
cur.execute("create table %s(id int, radius int)" % radius_table)
cur.execute("insert into %s select src_id, 0 from %s group by src_id"

cur.execute("drop index if exists edge_src_index")
cur.execute("drop index if exists edge_dst_index")
cur.execute("drop index if exists vertex_index")
cur.execute("drop index if exists tmp_index")
cur.execute("create index edge_src_index on %s (src_id)" % tmp_edge)
cur.execute("create index edge_dst_index on %s(dst_id)" % tmp_edge)

conn.commit()

assign_fm(conn, vertex_table, tmp_edge, s)
cur.execute("insert into %s select * from %s"  % (tmp_table, vertex_t
cur.execute("create index vertex_index on %s (id)" % vertex_table)
cur.execute("create index tmp_index on %s (id)" % tmp_table)
conn.commit()
print "initialized.."
max_iteration = 256
for i in range(max_iteration):
        print "iteration %d" % i
        if (i != 0):
                # cur.execute("delete from %s" % vertex_table)
                # cur.execute("insert into %s select * from %s" % (ve
                cur.execute("update %s as A set fm = B.fm from %s as
                conn.commit()
        update_bitstring(conn, tmp_edge, vertex_table, tmp_table)
        if (is_stablized(conn, vertex_table, tmp_table)):
                break
        cur.execute("insert into %s (select A.id, %d from %s as A, %s

cur.execute("insert into %s select id, max(radius) from %s group by i
cur.execute("insert into %s select distinct dst_id, 0 from %s except
conn.commit()
summarize(conn, radius_table, dataset)
drop_if_exists(conn, tmp_table)
drop_if_exists(conn, vertex_table)
drop_if_exists(conn, tmp_edge)
drop_if_exists(conn, hop_table)
cur.close()
```

```
        conn.close()
```

A.6.26 ../src/radius/radius_test.py

```
import psycopg2
from radius import radius
from functions import init_udf
import sys

conn = psycopg2.connect(database="mydb", host="127.0.0.1")

if __name__ == '__main__':
        init_udf(conn)
        compute_radius(conn, sys.argv[1], sys.argv[2])
```

A.6.27 ../src/radius/README

```
Weakly connected components.
```

A.6.28 ../src/radius/utils/agg_bit_or.sql

```
CREATE AGGREGATE agg_bit_or(bit[])
(
        sfunc = bit_or,
        stype = bit[]
);
```

A.6.29 ../src/radius/utils/bit_or.sql

```
create or replace function bit_or(BIT[], BIT[])
returns BIT[]
AS
$$
declare
bs1 ALIAS for $1;
bs2 ALIAS for $2;
retval BIT(16)[];
BEGIN
        for i in array_lower(bs1, 1)..array_upper(bs2, 1) LOOP
                retval[i] := bs1[i] | bs2[i];
        end loop;
        return retval;
END;
$$
language plpgsql
STRICT
```

A.6.30 ../src/radius/utils/fm_assin.sql

```
create or replace function fm_assign(int)
returns BIT[]
AS
$$
declare
dim ALIAS for $1;
retVal bit(16)[];
bitStr bit(16);
rndNum numeric;
index int;
BEGIN
        for i in 1..dim LOOP
                retVal[i] := 0::bit(16);
                select random() into rndNum;
                select floor(log(0.5, rndNum)) into index;
                if index >= 16 then
                        index := 15
                end
                select set_bit(retVal[i], index, 1) into bitStr;
                retVal[i] := bitStr;
                raise notice '% % %',bitStr, rndNum, index;
        end loop;
        return retVal;
END;
$$
language plpgsql
```

A.6.31 ../src/radius/utils/fm_size.sql

```
create or replace function fm_size(bit[])
returns numeric
AS
$$
declare
size numeric;
phi numeric := 0.77351;
bit_arr ALIAS for $1;
r int := 0;
k int := 0;
BEGIN
        for i in array_lower(bit_arr, 1)..array_upper(bit_arr, 1) loop
                r := r + lsb(bit_arr[i]);
```

```
            k = k + 1;
        end loop;
        raise notice '%', r;
        select power(2, 1.0 * r/k) / phi into size;
        return size;
END;
$$
language plpgsql;
```

A.6.32 ../src/radius/utils/functions.py

```
import psycopg2

# user defined functions
def create_bit_or(conn):
        func_def = \
                '''
                create or replace function bit_or(BIT[], BIT[])
                returns BIT[]
                AS
                $$
                declare
                bs1 ALIAS for $1;
                bs2 ALIAS for $2;
                retval BIT(32)[];
                BEGIN
                        for i in array_lower(bs1, 1)..array_upper(bs2, 1) LOC
                                retval[i] := bs1[i] | bs2[i];
                        end loop;
                        return retval;
                END;
                $$
                language plpgsql
                STRICT
                '''
        create_function(conn ,func_def)

def create_agg_bit_or(conn):
        cur = conn.cursor()
        cur.execute("drop aggregate if exists agg_bit_or(bit[])")
        func_def = \
        '''
        CREATE AGGREGATE agg_bit_or(bit[])
```

```
        (
                sfunc = bit_or,
                stype = bit[]
        );
        '''
        create_function(conn ,func_def)

def create_fm_assign(conn):
        func_def = \
        """
        create or replace function fm_assign(int)
        returns BIT[]
        AS
        $$
        declare
        dim ALIAS for $1;
        retVal bit(32)[];
        bitStr bit(32);
        rndNum numeric;
        index int;
        BEGIN
                for i in 1..dim LOOP
                        retVal[i] := 0::bit(32);
                        select random() into rndNum;
                        select floor(log(0.5, rndNum)) into index;
                        select set_bit(retVal[i], index, 1) into bitStr;
                        retVal[i] := bitStr;
                        raise notice '% % %',bitStr, rndNum, index;
                end loop;
                return retVal;
        END;
        $$
        language plpgsql
        """
        create_function(conn ,func_def)

def create_fm_size(conn):
        func_def = \
        """
        create or replace function fm_size(bit[])
        returns numeric
        AS
```

```
        $$
        declare
        size numeric;
        phi numeric := 0.77351;
        bit_arr ALIAS for $1;
        r int := 0;
        k int := 0;
        BEGIN
                for i in array_lower(bit_arr, 1)..array_upper(bit_arr, 1) loo
                        r := r + msb(bit_arr[i]);
                        k = k + 1;
                end loop;
                raise notice '%', r;
                select power(2, 1.0 * r/k) / phi into size;
                return size;
        END;
        $$
        language plpgsql;
        """
        create_function(conn ,func_def)

def create_msb(conn):
        func_def = \
        """
        create or replace function msb(bit)
        returns int
        AS
        $$
        declare
        len int;
        bitStr alias for $1;
        tmpBit int;

        BEGIN
                select length(bitStr) into len;
                for i in 1..len LOOP
                        select get_bit(bitStr, len - i) into tmpBit;
                        if tmpBit = 1 then
                                return len - i;
                        end if;
                end loop;
        END;
```

```
        $$
        language plpgsql
        """
        create_function(conn ,func_def)


def create_function(conn, func_def):
        cur = conn.cursor()
        cur.execute(func_def)
        cur.close()
        conn.commit()

def init_user_defined_funcs(conn):
        create_bit_or(conn)
        create_agg_bit_or(conn)
        create_msb(conn)
        create_fm_size(conn)
        create_fm_assign(conn)

if __name__ == '__main__':
        conn = psycopg2.connect(database="mydb", host="127.0.0.1")
        init_user_defined_funcs()
```

A.6.33 ../src/radius/utils/lsb.sql

```
create or replace function lsb(bit)
returns int
AS
$$
declare
len int;
bitStr alias for $1;
tmpBit int;

BEGIN
        select length(bitStr) into len;
        for i in 1..len LOOP
                select get_bit(bitStr, len - i - 1) into tmpBit;
                if tmpBit = 1 then
                        return i - 1;
                end if;
        end loop;
END;
```

```
$$
language plpgsql
```

A.6.34 ../src/spath/dijkstra.py

```
from common.basic_operation import *

select_id_query = '''\
select distinct nid \
from (\
    select from_id from %s \
    union\
    select to_id from %s\
    )\
as nid
'''

def dijkstra(start, graph, result, stamp, conn):
    """ dijksrta algorithm """
    # initilize table
    # 1. Distance (nodeid, distance, mark)
    # 2. Graph (from_id, to_id, value)
    cur = conn.cursor()
    distance = "Distance_%s" % stamp
    drop_if_exists(distance, conn)
    drop_if_exists(result, conn)
    cur.execute("create table %s (nodeid int, distance real, mark boolean)" %
    cur.execute("create table %s (nodeid int, distance real, mark boolean)" %
    cur.execute(select_id_query % (graph, graph))
    ids = cur.fetchall()
    # initialize all node's distance
    for i in ids:
        cur.execute("insert into %s values (%s, %s, '%s')" % (distance, i[0],
    # make start point best
    cur.execute("update %s set distance = 0 where nodeid = %s" % (distance, s
    cur.execute("create index nodeindex on %s (nodeid)" % (distance))
    for i in range(len(ids)):
        cur.execute("select nodeid, distance from %s where mark = 'f' order b
        candidate = cur.fetchone()
        if candidate[1] == 9999999999:
            break
        if i % 1000 == 0:
            print i, candidate[1]
```

```
        cur.execute("update %s set mark = 't' where nodeid = %s" % (distance,
        cur.execute("select to_id, value from %s,%s where from_id = %s and nc
        neighbours = cur.fetchall()
        # update neighbours
        for nbr in neighbours:
            cur.execute("update %s set distance = %s where distance > %s and
    assign_to(distance, result, conn)
    drop_if_exists(distance, conn)
    print "Result is in %s" % result
    conn.commit()
```

A.6.35 ../src/test_bp.py

```
import psycopg2
import unittest
from bp.bp import *
from common.util import *

class BeliefPropagationTest(unittest.TestCase):
        def setUp(self):
                self.conn = psycopg2.connect(database="mydb", host="127.0.0.1

        def tearDown(self):
                pass

        #@unittest.skip("")
        def test_amazon(self):
                """http://snap.stanford.edu/data/com-Amazon.html"""
                data_file = "../data/amazon.txt"
                edge_table = "amazon"
                dataset = "amazon"
                load_unweighted_graph(edge_table, data_file, True, self.conn)
                print "amazon.."
                compute_bp(self.conn, edge_table, dataset)

        #@unittest.skip("")
        def test_soc_sign_epinions(self):
                """http://snap.stanford.edu/data/soc-sign-epinions.html"""
                data_file = "../data/soc-sign-epinions.txt"
                edge_table = "soc_sign_epinions"
                dataset = "soc_sign_epinions"
                load_weighted_graph(edge_table, data_file, True, self.conn)
                print "soc_sign_epinions.."
```

```
        compute_bp(self.conn, edge_table, dataset)

    #@unittest.skip("")
    def test_email_EuAll(self):
        """http://snap.stanford.edu/data/email-EuAll.html"""
        data_file = "../data/email-EuAll.txt"
        edge_table = "email_EuAll"
        dataset = "email_EuAll"
        load_unweighted_graph(edge_table, data_file, True, self.conn)
        print "email_EuAll.."
        compute_bp(self.conn, edge_table, dataset)


    #@unittest.skip("")
    def test_web_google(self):
        """http://snap.stanford.edu/data/web-Google.html"""
        data_file = "../data/web_google.txt"
        edge_table = "web_google"
        dataset = "web_google"
        load_unweighted_graph(edge_table, data_file, True, self.conn)
        print "web google.."
        compute_bp(self.conn, edge_table, dataset)

    #@unittest.skip("")
    def test_youtube(self):
        """http://snap.stanford.edu/data/com-Youtube.html"""
        data_file = "../data/youtube.txt"
        edge_table = "youtube"
        dataset = "youtube"
        load_unweighted_graph(edge_table, data_file, True, self.conn)
        print "youtube.."
        compute_bp(self.conn, edge_table, dataset)

    #@unittest.skip("")
    def test_dblp(self):
        """dblp"""
        data_file = "../data/dblp.txt"
        edge_table = "dblp"
        dataset = "dblp"
        load_unweighted_graph(edge_table, data_file, True, self.conn)
        print "dblp.."
        compute_bp(self.conn, edge_table, dataset)
```

```
        @unittest.skip("")
        def test_synthetic(self):
                """test_synthetic"""
                data_file = "../data/synthetic_bp.txt"
                edge_table = "synthetic"
                dataset = "synthetic"
                load_weighted_graph(edge_table, data_file, False, self.conn,
                print "synthetic.."
                compute_bp(self.conn, edge_table, dataset, True)

        @unittest.skip("")
        def test_advogato(self):
                """test_advogato"""
                data_file = "../data/advogato.txt"
                edge_table = "advogato"
                dataset = "advogato"
                load_weighted_graph(edge_table, data_file, True, self.conn, "
                print "advogato.."
                compute_bp(self.conn, edge_table, dataset, True)
```

A.6.36 ../src/test_connected_component.py

```
import psycopg2
import unittest
from cc.cc import *
from common.util import *

class ConnectedComponentTest(unittest.TestCase):
        def setUp(self):
                self.conn = psycopg2.connect(database="mydb", host="127.0.0.1

        def tearDown(self):
                pass

        #@unittest.skip("")
        def test_trec_wt10(self):
                """http://konect.uni-koblenz.de/networks/trec-wt10g"""
                data_file = "../data/trec_wt10.txt"
                edge_table = "trec_wt10"
                target_table = "cc_trec_wt10"
                load_weighted_graph(edge_table, data_file, False, self.conn,
                print "trec_wt10.."
```

```
        compute_cc(self.conn, edge_table, target_table)

    @unittest.skip("")
    def test_youtube(self):
        """http://snap.stanford.edu/data/com-Youtube.html"""
        data_file = "../data/youtube.txt"
        edge_table = "youtube"
        target_table = "cc_youtube"
        load_unweighted_graph(edge_table, data_file, False, self.conn
        print "youtube.."
        compute_cc(self.conn, edge_table, target_table)

    @unittest.skip("")
    def test_pokec(self):
        """"http://snap.stanford.edu/data/soc-pokec.html"""
        data_file = "../data/pokec.txt"
        edge_table = "pokec"
        target_table = "cc_pokec"
        load_unweighted_graph(edge_table, data_file, False, self.conn
        print "pokec.."
        compute_cc(self.conn, edge_table, target_table)

    @unittest.skip("")
    def test_web_google(self):
        """http://snap.stanford.edu/data/web-Google.html"""
        data_file = "../data/web_google.txt"
        edge_table = "web_google"
        target_table = "cc_web_google"
        load_unweighted_graph(edge_table, data_file, False, self.conn
        print "web google.."
        compute_cc(self.conn, edge_table, target_table)

    @unittest.skip("")
    def test_amazon(self):
        """http://snap.stanford.edu/data/com-Amazon.html"""
        data_file = "../data/amazon.txt"
        edge_table = "amazon"
        target_table = "cc_amazon"
        load_unweighted_graph(edge_table, data_file, False, self.conn
        print "amazon.."
        compute_cc(self.conn, edge_table, target_table)
```

```
    @unittest.skip("")
    def test_email_EuAll(self):
        """http://snap.stanford.edu/data/email-EuAll.html"""
        data_file = "../data/email-EuAll.txt"
        edge_table = "email_EuAll"
        target_table = "cc_email_EuAll"
        load_unweighted_graph(edge_table, data_file, False, self.conn
        print "email_EuAll.."
        compute_cc(self.conn, edge_table, target_table)


    @unittest.skip("")
    def test_wiki_talk(self):
        """http://snap.stanford.edu/data/wiki-Talk.html"""
        data_file = "../data/wiki-talk.txt"
        edge_table = "wiki_talk"
        target_table = "cc_wiki_talk"
        load_unweighted_graph(edge_table, data_file, False, self.conn
        print "wiki_talk.."
        compute_cc(self.conn, edge_table, target_table)


    @unittest.skip("")
    def test_soc_sign_epinions(self):
        """http://snap.stanford.edu/data/soc-sign-epinions.html"""
        data_file = "../data/soc-sign-epinions.txt"
        edge_table = "soc_sign_epinions"
        target_table = "cc_soc_sign_epinions"
        load_weighted_graph(edge_table, data_file, False, self.conn)
        print "soc_sign_epinions.."
        compute_cc(self.conn, edge_table, target_table)
```

A.6.37 ../src/test_degree_distribution.py

```
import sys
import time
import psycopg2
import unittest

from common.util import *
from ddis.ddis import *

class DegreeDistributionTest(unittest.TestCase):
```

```
    def setUp(self):
        self.conn = psycopg2.connect(database="mydb", host="127.0.0.1", user=

    def tearDown(self):
        pass

    @unittest.skip("skip roadnet ca")
    def test_roadnet_ca(self):
        """http://snap.stanford.edu/data/roadNet-CA.html"""
        # undirected, no reverse
        data_file = "data/roadNet-CA.txt"
        tbl_name = "task1_roadnetca"
        load_undirected_graph_into_table(tbl_name, data_file, False, self.con
        print "roadNet-CA....."
        undirect_dis(tbl_name, 'task1_roadnetca_result', self.conn)

    @unittest.skip("")
    def test_wiki_talk(self):
        """http://snap.stanford.edu/data/wiki-Talk.html"""
        # directed
        data_file = "data/wiki-Talk.txt"
        tbl_name = "task1_wikitalk"
        result_tbl = "task1_wikitalk_result_"
        load_undirected_graph_into_table(tbl_name, data_file, False, self.con
        indis(tbl_name, result_tbl+"in", self.conn)
        outdis(tbl_name, result_tbl+"out", self.conn)

    @unittest.skip("")
    def test_roadnet_pa(self):
        """http://snap.stanford.edu/data/roadNet-PA.html"""
        # undirected, no reverse
        data_file = "data/roadNet-PA.txt"
        tbl_name = "task1_roadnetpa"
        load_undirected_graph_into_table(tbl_name, data_file, False, self.con
        print "roadNet-PA....."
        undirect_dis(tbl_name, 'task1_roadnetpa_result', self.conn)

    @unittest.skip("")
    def test_roadnet_tx(self):
        """http://snap.stanford.edu/data/roadNet-TX.html"""
        # undirected, no reverse
        data_file = "data/roadNet-TX.txt"
```

```
        tbl_name = "task1_roadnettx"
        load_undirected_graph_into_table(tbl_name, data_file, False, self.con
        print "roadNet-TX....."
        undirect_dis(tbl_name, 'task1_roadnettx_result', self.conn)

    @unittest.skip("")
    def test_youtube(self):
        """http://snap.stanford.edu/data/com-Youtube.html"""
        # undirected, need reverse
        data_file = "data/com-youtube.ungraph.txt"
        tbl_name = "task1_youtube"
        result_tbl = "task1_youtube_result"
        load_undirected_graph_into_table(tbl_name, data_file, True, self.conn
        undirect_dis(tbl_name, result_tbl, self.conn)
```

A.6.38 ../src/test_eigenvalue.py

```
import sys
import time
import unittest
import psycopg2
from common.basic_operation import *
from common.util import *
from eigenvalue.lanczos import *

class EigenvalueTest(unittest.TestCase):
    def setUp(self):
        self.conn = psycopg2.connect(database="mydb", host="127.0.0.1", user=

    def tearDown(self):
        pass

    def test_ucidata(self):
        data_file = "data/ucidata-zachary.txt"
        gfile = "task5_ucidata"
        cur = self.conn.cursor()
        drop_if_exists(gfile, self.conn)
        cur.execute("create table %s(row int, col int, value real DEFAULT 1)"
        cur.copy_from(open(data_file), gfile, columns=('row', 'col'), sep=" "
        cur.execute("update %s set col=col-1, row=row-1" % gfile)
        self.conn.commit()
        print "Ucidata builded ..."
        reverse_matrix(gfile, self.conn)
```

```
        b = 'b'
        create_vector_or_matrix(b, self.conn)
        cur.execute("select max(col) from %s" % gfile)
        dim = cur.fetchone()[0] + 1
        print "Dimension is %s" % dim
        for i in range(dim):
            cur.execute("insert into %s values (%s, %s, %s)" % (b, i, 0, 1.0
        lanczos(gfile, b, dim, 20, self.conn)
        drop_if_exists(b, self.conn)

    @unittest.skip("")
    def test_dblp_cite(self):
        """http://konect.uni-koblenz.de/networks/dblp-cite"""
        data_file = "data/dblp-cite.txt"
        gfile = "task5_dblpcite"
        cur = self.conn.cursor()
        drop_if_exists(gfile, self.conn)
        cur.execute("create table %s(row int, col int, value real DEFAULT 1)"
        cur.copy_from(open(data_file), gfile, columns=('row', 'col'), sep=" "
        cur.execute("update %s set col=col-1, row=row-1" % gfile)
        self.conn.commit()
        print "DBLP builded ..."
        reverse_matrix(gfile, self.conn)
        b = 'b'
        create_vector_or_matrix(b, self.conn)
        cur.execute("select max(col) from %s" % gfile)
        dim = cur.fetchone()[0] + 1
        print "Dimension is %s" % dim
        for i in range(dim):
            cur.execute("insert into %s values (%s, %s, %s)" % (b, i, 0, 1.0
        lanczos(gfile, b, dim, 20, self.conn)
        drop_if_exists(b, self.conn)

    @unittest.skip("")
    def test_youtube(self):
        """http://snap.stanford.edu/data/com-Youtube.html"""
        data_file = "data/com-youtube.ungraph.txt"
        gfile = "task5_youtube"
        cur = self.conn.cursor()
        drop_if_exists(gfile, self.conn)
        cur.execute("create table %s(row int, col int, value real DEFAULT 1)"
        cur.copy_from(open(data_file), gfile, columns=('row', 'col'), sep="\t
```

```
        cur.execute("update %s set col=col-1, row=row-1" % gfile)
        self.conn.commit()
        print "Youtube builded ..."
        reverse_matrix(gfile, self.conn)
        b = 'b'
        create_vector_or_matrix(b, self.conn)
        cur.execute("select max(col) from %s" % gfile)
        dim = cur.fetchone()[0] + 1
        print "Dimension is %s" % dim
        for i in range(dim):
            cur.execute("insert into %s values (%s, %s, %s)" % (b, i, 0, 1.0
        lanczos(gfile, b, dim, 20, self.conn)
        drop_if_exists(b, self.conn)

    @unittest.skip("")
    def test_uci_gamma(self):
        """http://konect.uni-koblenz.de/networks/ucidata-gama"""
        data_file = "data/ucidata-gama.txt"
        gfile = "task5_ucigama"
        cur = self.conn.cursor()
        drop_if_exists(gfile, self.conn)
        cur.execute("create table %s(row int, col int, value real DEFAULT 1)"
        cur.execute("create table tmppppp(row int, col int, value real DEFAUL
        cur.copy_from(open(data_file), "tmppppp", columns=('row', 'col', 'val
        cur.execute("insert into %s select row, col from tmppppp" % gfile)
        drop_if_exists("tmppppp", self.conn)
        cur.execute("update %s set col=col-1, row=row-1" % gfile)
        self.conn.commit()
        print "Uci gama builded ..."
        reverse_matrix(gfile, self.conn)
        b = 'b'
        create_vector_or_matrix(b, self.conn)
        cur.execute("select max(col) from %s" % gfile)
        dim = cur.fetchone()[0] + 1
        print "Dimension is %s" % dim
        for i in range(dim):
            cur.execute("insert into %s values (%s, %s, %s)" % (b, i, 0, 1.0
        lanczos(gfile, b, dim, 10, self.conn)
        drop_if_exists(b, self.conn)
```

A.6.39 ../src/test_pagerank.py

```
import sys
```

```
import time
import unittest
import psycopg2
from common.basic_operation import *
from common.util import *
from pagerank.pagerank import *

class PagerankTest(unittest.TestCase):
    def setUp(self):
        self.conn = psycopg2.connect(database="mydb", host="127.0.0.1", user=

    def tearDown(self):
        pass


    def test_dummy_pagerank(self):
        tbl_name = "task2_dummy"
        result_tbl = "task2_dummy_result"
        drop_if_exists(tbl_name, self.conn)
        drop_if_exists(result_tbl, self.conn)
        data = [[0,0,1,0,0,0,0],
                [0,1,1,0,0,0,0],
                [1,0,1,1,0,0,0],
                [0,0,0,1,1,0,0],
                [0,0,0,0,0,0,1],
                [0,0,0,0,0,1,1],
                [0,0,0,1,1,0,1]]
        cur = self.conn.cursor()
        cur.execute("create table %s (from_id int, to_id int)" % tbl_name)
        for i in range(len(data)):
            for j in range(len(data)):
                if data[i][j] == 1:
                    cur.execute("insert into %s values (%s, %s)" % (tbl_name,
        print "Dummy build"
        calculatepagerank(tbl_name, result_tbl, self.conn)
        normalize_pagerank(result_tbl, self.conn)
        cur = self.conn.cursor()
        cur.execute("select * from %s order by node_id" % result_tbl)
        r = cur.fetchall()
        print r

    @unittest.skip("")
```

```
def test_google_webgraph(self):
    """http://snap.stanford.edu/data/web-Google.html"""
    data_file = "data/web-Google.txt"
    tbl_name = "task2_googleweb"
    result_tbl = "task2_googleweb_result"
    load_undirected_graph_into_table(tbl_name, data_file, False, self.con
    print "Google webgraph....."
    calculatepagerank(tbl_name, result_tbl, self.conn)

@unittest.skip("")
def test_berkeley_stanford(self):
    """http://snap.stanford.edu/data/web-BerkStan.html"""
    data_file = "data/web-BerkStan.txt"
    tbl_name = "task2_berkstan"
    result_tbl = "task2_berkstan_result"
    load_undirected_graph_into_table(tbl_name, data_file, False, self.con
    print "Berkeley stanford....."
    calculatepagerank(tbl_name, result_tbl, self.conn)

@unittest.skip("")
def test_web_stanford(self):
    """http://snap.stanford.edu/data/web-Stanford.html"""
    data_file = "data/web-Stanford.txt"
    tbl_name = "task2_webstanford"
    result_tbl = "task2_webstanford_result"
    load_undirected_graph_into_table(tbl_name, data_file, False, self.con
    print "Web Stanford....."
    calculatepagerank(tbl_name, result_tbl, self.conn)

@unittest.skip("")
def test_amazon_purchase(self):
    """http://snap.stanford.edu/data/com-Amazon.html"""
    # undirected, need reverse
    data_file = "data/com-amazon.ungraph.txt"
    tbl_name = "task2_amazon_product"
    result_tbl = "task2_amazon_product_result"
    load_undirected_graph_into_table(tbl_name, data_file, True, self.conn
    print "Amazone product....."
    calculatepagerank(tbl_name, result_tbl, self.conn)

@unittest.skip("")
def test_slashdot0902(self):
```

```
        """http://snap.stanford.edu/data/soc-Slashdot0902.html"""
        data_file = "data/soc-Slashdot0902.txt"
        tbl_name = "task2_slashdot0902"
        result_tbl = "task2_slashdot0902_result"
        load_undirected_graph_into_table(tbl_name, data_file, False, self.con
        print "Slashdot0902....."
        calculatepagerank(tbl_name, result_tbl, self.conn)

    @unittest.skip("")
    def test_enron_email(self):
        """http://snap.stanford.edu/data/email-Enron.html"""
        data_file = "data/email-Enron.txt"
        tbl_name = "task2_enronmail"
        result_tbl = "task2_enronmail_result"
        load_undirected_graph_into_table(tbl_name, data_file, False, self.con
        print "Enron-mail....."
        calculatepagerank(tbl_name, result_tbl, self.conn)

    @unittest.skip("")
    def test_wiki_vote(self):
        """http://snap.stanford.edu/data/wiki-Vote.html"""
        # directed
        data_file = "data/wiki-Vote.txt"
        tbl_name = "task2_wikivote"
        result_tbl = "task2_wikivote_result"
        load_undirected_graph_into_table(tbl_name, data_file, False, self.con
        print "Wiki-cote....."
        calculatepagerank(tbl_name, result_tbl, self.conn)

    @unittest.skip("")
    def test_roadnet_ca(self):
        """http://snap.stanford.edu/data/roadNet-CA.html"""
        # undirected, no reverse
        data_file = "data/roadNet-CA.txt"
        tbl_name = "task2_roadnetca"
        result_tbl = "task2_roadnetca_result"
        load_undirected_graph_into_table(tbl_name, data_file, False, self.con
        print "roadNet-CA....."
        calculatepagerank(tbl_name, result_tbl, self.conn)


    @unittest.skip("")
```

```
    def test_wiki_talk(self):
        """http://snap.stanford.edu/data/wiki-Talk.html"""
        # directed
        data_file = "data/wiki-Talk.txt"
        tbl_name = "task2_wikitalk"
        result_tbl = "task2_wikitalk_result"
        load_undirected_graph_into_table(tbl_name, data_file, False, self.con
        calculatepagerank(tbl_name, result_tbl, self.conn)

    @unittest.skip("")
    def test_roadnet_pa(self):
        """http://snap.stanford.edu/data/roadNet-PA.html"""
        # undirected, no reverse
        data_file = "data/roadNet-PA.txt"
        tbl_name = "task2_roadnetpa"
        result_tbl = "task2_roadnetpa_result"
        load_undirected_graph_into_table(tbl_name, data_file, False, self.con
        print "roadNet-PA....."
        calculatepagerank(tbl_name, result_tbl, self.conn)

    @unittest.skip("")
    def test_roadnet_tx(self):
        """http://snap.stanford.edu/data/roadNet-TX.html"""
        # undirected, no reverse
        data_file = "data/roadNet-TX.txt"
        tbl_name = "task2_roadnettx"
        result_tbl = "task2_roadnettx_result"
        load_undirected_graph_into_table(tbl_name, data_file, False, self.con
        print "roadNet-TX....."
        calculatepagerank(tbl_name, result_tbl, self.conn)

    @unittest.skip("")
    def test_youtube(self):
        """http://snap.stanford.edu/data/com-Youtube.html"""
        # undirected, need reverse
        data_file = "data/com-youtube.ungraph.txt"
        tbl_name = "task2_youtube"
        result_tbl = "task2_youtube_result"
        load_undirected_graph_into_table(tbl_name, data_file, True, self.conn
        calculatepagerank(tbl_name, result_tbl, self.conn)
```

A.6.40 ../src/test_radius.py

```
import psycopg2
import unittest
from radius.radius import *
from radius.functions import *
from common.util import *

class RadiusTest(unittest.TestCase):
        def setUp(self):
                self.conn = psycopg2.connect(database="mydb", host="127.0.0.1
                init_udf(self.conn)

        def tearDown(self):
                pass

        @unittest.skip("")
        def test_amazon(self):
                """http://snap.stanford.edu/data/com-Amazon.html"""
                data_file = "../data/amazon.txt"
                edge_table = "amazon"
                dataset = "amazon"
                load_unweighted_graph(edge_table, data_file, True, self.conn)
                print "amazon.."
                compute_radius(self.conn, edge_table, dataset, 4)

        @unittest.skip("")
        def test_soc_sign_epinions(self):
                """http://snap.stanford.edu/data/soc-sign-epinions.html"""
                data_file = "../data/soc-sign-epinions.txt"
                edge_table = "soc_sign_epinions"
                dataset = "soc_sign_epinions"
                load_weighted_graph(edge_table, data_file, True, self.conn)
                print "soc_sign_epinions.."
                compute_radius(self.conn, edge_table, dataset, 1)

        @unittest.skip("")
        def test_email_EuAll(self):
                """http://snap.stanford.edu/data/email-EuAll.html"""
                data_file = "../data/email-EuAll.txt"
                edge_table = "email_EuAll"
                dataset = "email_EuAll"
                load_unweighted_graph(edge_table, data_file, True, self.conn)
                print "email_EuAll.."
```

```
        compute_radius(self.conn, edge_table, dataset, 8)

    #@unittest.skip("")
    def test_web_google(self):
        """http://snap.stanford.edu/data/web-Google.html"""
        data_file = "../data/web_google.txt"
        edge_table = "web_google"
        dataset = "web_google"
        load_unweighted_graph(edge_table, data_file, True, self.conn)
        print "web google.."
        compute_radius(self.conn, edge_table, dataset, 1)

    @unittest.skip("")
    def test_youtube(self):
        """http://snap.stanford.edu/data/com-Youtube.html"""
        data_file = "../data/youtube.txt"
        edge_table = "youtube"
        dataset = "youtube"
        load_unweighted_graph(edge_table, data_file, True, self.conn)
        print "youtube.."
        compute_radius(self.conn, edge_table, dataset, 1)

    @unittest.skip("")
    def test_dblp(self):
        """dblp"""
        data_file = "../data/dblp.txt"
        edge_table = "dblp"
        dataset = "dblp"
        load_unweighted_graph(edge_table, data_file, True, self.conn)
        print "dblp.."
        compute_radius(self.conn, edge_table, dataset, 1)

    @unittest.skip("")
    def test_synthetic(self):
        """test_synthetic"""
        data_file = "../data/synthetic.txt"
        edge_table = "synthetic"
        dataset = "synthetic"
        load_unweighted_graph(edge_table, data_file, True, self.conn,
        print "synthetic.."
        compute_radius(self.conn, edge_table, dataset, 32)
```

A.6.41 ../src/test_shortest_path.py

```
import sys
import time
import psycopg2
import unittest
from common.basic_operation import *
from spath.dijkstra import *
from graph_generator import *

class ShortestPathTest(unittest.TestCase):
    def setUp(self):
        self.conn = psycopg2.connect(database="mydb", host="127.0.0.1")
        # self.conn = psycopg2.connect(database="mydb", host="127.0.0.1", use

    def tearDown(self):
        pass

    @unittest.skip("")
    def test_dummydata(self):
        cur = self.conn.cursor()
        gname = "spath_graph"
        graph = [['1', '2', '1.0'], \
                 ['2', '3', '1.0'], \
                 ['1', '3', '1.0'], \
                 ['3', '4', '0.5'], \
                 ['3', '6', '1.6'], \
                 ['2', '6', '0.3']]
        drop_if_exists(gname, self.conn)
        cur.execute("create table %s (from_id int, to_id int, value real)" %
        for edge in graph:
            cur.execute("insert into %s values (%s, %s, %s)" % (gname, edge[0
        dijkstra("1", gname, "spath_result", "spath", self.conn)
        cur = self.conn.cursor()
        cur.execute("select * from spath_result")
        print "Shortest Path result"
        for r in cur.fetchall():
            print r

    def test_runtime(self):
        cur = self.conn.cursor()
        gname = "spath_graph"
        generate_directed_graph(gname, 100, self.conn)
```

```
        print "graph builded"
        dijkstra("0", gname, "spath_result", "spath", self.conn)
        cur = self.conn.cursor()
        cur.execute("select * from spath_result")
        print "Shortest Path result"
```

A.6.42 ../src/test_triangle.py

```
import psycopg2
import unittest
import logging
from common.basic_operation import *
from triangle.tria import *

class TriangleTest(unittest.TestCase):
    def setUp(self):
        self.conn = psycopg2.connect(database="mydb", host="127.0.0.1", user=

    def tearDown(self):
        pass

    def test_youtube(self):
        data_file = "data/com-youtube.ungraph.txt"
        gfile = "task7_youtube_graph"
        cur = self.conn.cursor()
        drop_if_exists(gfile, self.conn)
        cur.execute("create table %s(row int, col int, value real DEFAULT 1)"
        cur.copy_from(open(data_file), gfile, columns=('row', 'col'))
        print "Youtube graph builded ..."
        reverse_matrix(gfile, self.conn)
        r = count_triangle(gfile, self.conn)
        print "There are %s triangles in [com-youtube.ungraph.txt]." % r
        drop_if_exists(gfile, self.conn)

    @unittest.skip("")
    def test_wikivote(self):
        """http://snap.stanford.edu/data/wiki-Vote.html"""
        data_file = "data/wiki-Vote.txt"
        gfile = "task7_wiki_vote"
        cur = self.conn.cursor()
        drop_if_exists(gfile, self.conn)
        cur.execute("create table %s(row int, col int, value real DEFAULT 1)"
        cur.copy_from(open(data_file), gfile, columns=('row', 'col'))
```

```
        print "Wiki vote builded ..."
        reverse_matrix(gfile, self.conn)
        r = count_triangle(gfile, self.conn)
        print "There are %s triangles." % r
        drop_if_exists(gfile, self.conn)

    @unittest.skip("")
    def test_enron_email(self):
        """http://snap.stanford.edu/data/email-Enron.html"""
        data_file = "data/email-Enron.txt"
        gfile = "task7_enron_email"
        cur = self.conn.cursor()
        drop_if_exists(gfile, self.conn)
        cur.execute("create table %s(row int, col int, value real DEFAULT 1)"
        cur.copy_from(open(data_file), gfile, columns=('row', 'col'))
        print "Enron email builded ..."
        r = count_triangle(gfile, self.conn)
        print "There are %s triangles." % r
        drop_if_exists(gfile, self.conn)

    @unittest.skip("")
    def test_slashdot0902(self):
        """http://snap.stanford.edu/data/soc-Slashdot0902.html"""
        data_file = "data/soc-Slashdot0902.txt"
        gfile = "task7_slashdot0902"
        cur = self.conn.cursor()
        drop_if_exists(gfile, self.conn)
        cur.execute("create table %s(row int, col int, value real DEFAULT 1)"
        cur.copy_from(open(data_file), gfile, columns=('row', 'col'))
        print "Slashdot 0902 builded ..."
        r = count_triangle(gfile, self.conn)
        print "There are %s triangles." % r
        drop_if_exists(gfile, self.conn)

    @unittest.skip("")
    def test_amazon_purchase(self):
        """http://snap.stanford.edu/data/com-Amazon.html"""
        data_file = "data/com-amazon.ungraph.txt"
        gfile = "task7_amazon_purchase"
        cur = self.conn.cursor()
        drop_if_exists(gfile, self.conn)
        cur.execute("create table %s(row int, col int, value real DEFAULT 1)"
```

```
        cur.copy_from(open(data_file), gfile, columns=('row', 'col'))
        reverse_matrix(gfile, self.conn)
        print "Amazon builded ..."
        r = count_triangle(gfile, self.conn)
        print "There are %s triangles." % r
        drop_if_exists(gfile, self.conn)

    @unittest.skip("")
    def test_web_stanford(self):
        """http://snap.stanford.edu/data/web-Stanford.html"""
        data_file = "data/web-Stanford.txt"
        gfile = "task7_web_stanford"
        cur = self.conn.cursor()
        drop_if_exists(gfile, self.conn)
        cur.execute("create table %s(row int, col int, value real DEFAULT 1)"
        cur.copy_from(open(data_file), gfile, columns=('row', 'col'))
        print "Web stanford builded ..."
        r = count_triangle(gfile, self.conn)
        print "There are %s triangles." % r
        drop_if_exists(gfile, self.conn)

    @unittest.skip("")
    def test_roadnet_pa(self):
        """http://snap.stanford.edu/data/roadNet-PA.html"""
        data_file = "data/roadNet-PA.txt"
        gfile = "task7_roadnet_pa"
        cur = self.conn.cursor()
        drop_if_exists(gfile, self.conn)
        cur.execute("create table %s(row int, col int, value real DEFAULT 1)"
        cur.copy_from(open(data_file), gfile, columns=('row', 'col'))
        print "Roadnet PA builded ..."
        r = count_triangle(gfile, self.conn)
        print "There are %s triangles." % r
        drop_if_exists(gfile, self.conn)
```

A.6.43 ../src/test_triangle_local.py

```
import psycopg2
import unittest
import logging
from common.basic_operation import *
from triangle.tria import *
```

```
class LocalTriangleTest(unittest.TestCase):
    def setUp(self):
        self.conn = psycopg2.connect(database="mydb", host="127.0.0.1", user=

    def tearDown(self):
        pass

    @unittest.skip("")
    def test_wikivote(self):
        """http://snap.stanford.edu/data/wiki-Vote.html"""
        data_file = "data/wiki-Vote.txt"
        gfile = "task7_wiki_vote"
        cur = self.conn.cursor()
        drop_if_exists(gfile, self.conn)
        cur.execute("create table %s(row int, col int, value real DEFAULT 1)"
        cur.copy_from(open(data_file), gfile, columns=('row', 'col'))
        print "Wiki vote builded ..."
        reverse_matrix(gfile, self.conn)
        # count local triangle
        print "calculating local triangles"
        count_local_triangle(gfile, self.conn)
        drop_if_exists(gfile, self.conn)

    @unittest.skip("")
    def test_enron_email(self):
        """http://snap.stanford.edu/data/email-Enron.html"""
        data_file = "data/email-Enron.txt"
        gfile = "task7_enron_email"
        cur = self.conn.cursor()
        drop_if_exists(gfile, self.conn)
        cur.execute("create table %s(row int, col int, value real DEFAULT 1)"
        cur.copy_from(open(data_file), gfile, columns=('row', 'col'))
        print "Enron email builded ..."
        count_local_triangle(gfile, self.conn)
        drop_if_exists(gfile, self.conn)

    @unittest.skip("")
    def test_slashdot0902(self):
        """http://snap.stanford.edu/data/soc-Slashdot0902.html"""
        data_file = "data/soc-Slashdot0902.txt"
        gfile = "task7_slashdot0902"
        cur = self.conn.cursor()
```

```
        drop_if_exists(gfile, self.conn)
        cur.execute("create table %s(row int, col int, value real DEFAULT 1)"
        cur.copy_from(open(data_file), gfile, columns=('row', 'col'))
        print "Slashdot 0902 builded ..."
        count_local_triangle(gfile, self.conn)
        drop_if_exists(gfile, self.conn)

    @unittest.skip("")
    def test_amazon_purchase(self):
        """http://snap.stanford.edu/data/com-Amazon.html"""
        data_file = "data/com-amazon.ungraph.txt"
        gfile = "task7_amazon_purchase"
        cur = self.conn.cursor()
        drop_if_exists(gfile, self.conn)
        cur.execute("create table %s(row int, col int, value real DEFAULT 1)"
        cur.copy_from(open(data_file), gfile, columns=('row', 'col'))
        reverse_matrix(gfile, self.conn)
        print "Amazon builded ..."
        count_local_triangle(gfile, self.conn)
        drop_if_exists(gfile, self.conn)

    def test_youtube(self):
        data_file = "data/com-youtube.ungraph.txt"
        gfile = "task7_youtube_graph"
        cur = self.conn.cursor()
        drop_if_exists(gfile, self.conn)
        cur.execute("create table %s(row int, col int, value real DEFAULT 1)"
        cur.copy_from(open(data_file), gfile, columns=('row', 'col'))
        print "Youtube graph builded ..."
        reverse_matrix(gfile, self.conn)
        count_local_triangle(gfile, self.conn)
        drop_if_exists(gfile, self.conn)
```

A.6.44 ../src/triangle/tria.py

```
from common.basic_operation import *
from eigenvalue.eigen_quodratic import *
from eigenvalue.lanczos import *

def count_triangle(tbl_name, conn):
    """ input is a matrix """
    tol = 0.1
    b = 'b'
```

```
    lim = 10
    cur = conn.cursor()
    tn = tbl_name
    print "create matrix..."
    create_vector_or_matrix(b, conn)
    print "Counting dimension..."
    calc_dim_query = "select max(row), max(col) from %s" % (tn)
    cur.execute(calc_dim_query)
    p = cur.fetchone()
    dim = max(p) + 1
    print "dimension is %s" % dim
    appro = min(dim, lim);
    for i in range(dim):
        cur.execute("insert into %s values (%s, %s, %s)" % (b, i, 0, 1.0 / fl
    print "init b..."
    lanczos(tn, b, dim, appro, conn)
    cur.execute("select power(value,3.0) from eigenval where row = col order
    r = cur.fetchall()
    print r
    s = 0.0001
    for i in range(len(r)):
        if r[i][0] < 0 or abs(r[i][0]) / s < tol:
            break
        s += r[i][0]
    drop_if_exists(tn, conn)
    drop_if_exists(b, conn)
    # drop_if_exists("eigenval", conn)
    # drop_if_exists("eigenvec", conn)
    return s / 6.0

def count_local_triangle(tbl_name, conn):
    """ input is a matrix """
    tol = 0.1
    b = 'b'
    lim = 10
    cur = conn.cursor()
    tn = tbl_name
    print "create matrix..."
    create_vector_or_matrix(b, conn)
    print "Counting dimension..."
    calc_dim_query = "select max(row), max(col) from %s" % (tn)
    cur.execute(calc_dim_query)
```

```
p = cur.fetchone()
dim = max(p) + 1
print "dimension is %s" % dim
appro = min(dim, lim);
for i in range(dim):
    cur.execute("insert into %s values (%s, %s, %s)" % (b, i, 0, 1.0 / fl
print "init b..."
lanczos(tn, b, dim, appro, conn)
ltriangle = "task7_local_triangle_count"
drop_if_exists(ltriangle, conn)
cur = conn.cursor()
cur.execute("create table %s (nid int, cnt int)" % ltriangle)
print "calculating the local triangle count for each node"
cur.execute("insert into %s select U.row, round(sum(power(U.value, 2.0) *
```

Contents