Carnegie Mellon University
15-826 Multimedia Databases and Data Mining
Fall 2013 - C. Faloutsos
Project 2: Graph Mining using SQL
Final Report

Emma R. Zhang
School of Computer Science
runyunz@andrew.cmu.edu(runyunz)

Fangyu Gao
School of Computer Science
fangyug@andrew.cmu.edu(fangyug)

November 28, 2013

Abstract

It is natural to expect performing data mining in RDBMS, where data stored. Can we perform graph mining directly in database, without moving data all round? Can we implement various graph query methods via SQL, so as to leverage the powerful support from RDBMS? Can we figure out a maverick graph, which has unusual features among a similar set of graph datasets? In this project, our goal is to implement a SQL version of graph mining algorithms, and perform broad spectrum graph mining on a variety of graph dataset with the support of RDBMS.

1 Introduction

The problem we want to solve is the following:

- GIVEN: a collection of graph datasets from various scenarios
- FIND: 'strange' datasets which have different behaviors versus the majority datasets
- to IMPLEMENT: a set of powerful graph mining methods in SQL on RDBMS, including *degree distribution, page rank, connected components, radius, eigenvalues/singular values, Belief Propagation, Count of Triangles, etc.*

Data, especially those containing valuable information, are stored in Database. Therefore, it is reasonable that people wish to perform data mining directly within RDBMS. In addition, RDBMS offers powerful support from all perspective, be it sophisticated data management mechanism, fine-tuned query optimization, and easy-to-use interface language: SQL. However, the challenge is that it is a unnatural way to implement graph query algorithms using ralational query. Therefore, our goal is to address this problem so that we can perform graph mining with full support of RDMS and find interesting patterns of various graph datasets.

The contributions of this project are the following:

- Our implementation of graph queries algorithms in SQL provide a native way for graph mining to take advantage of RDBMS
- Based on our implementated algorithm, we can perform broad spectrum graph mining on a variety of graph dataset efficiently

2 Survey

Next we list the papers that each member read, along with their summary and critique. Table 1 gives a list of common symbols we used.

Symbol	Definition
V	Vertices of graph
E	Edges of graph

Table 1: Symbols and definitions

2.1 Papers read by Emma Zhang

The first paper was the GBASE paper [5].

- *Main idea*: GBASE is a efficient, scalable platform on Hadoop designed for large-scale graph mining(e.g. perform real analysis on YahooWeb, one of the largest graphs in the world with 1.4 billion nodes, 6.6 bilion edges and 120GB in space).
  The overall framework of GBASE is composed of two stages. During the indexing stage, GBASE offers a novel graph storage method. By using a set of carefully-designed clustering, compression and blocking algorithms, the new method enjoys high performance in terms of space efficiency, running time, and machine scalability.
  During the query stage, GBASE unifies 11 types of graph query operations by matrix-vetor multiplication, including *degree distribution, PageRank, Random Walk with Restart, radius estimations, discovery of connected components, LineRank* as global queries, and *neighborhod, induced subgraph, Egonet, K-core, cross-edges and single source shortest*

distances as targeted queries. Compared with previous work(PEGASUS [7]), the contribution of GBASE is that 1) node-based and edge-based queries are unified; 2) the platform supports both adjacency matrix and incidence matrix.
In a nutshell, GBASE solves problem and achieves remarkable improvement in terms of storage, algorithms and query optimization.

- *Use for our project*: In addition to the Matrix-Vector implementation on Hadoop, the author also provided the SQL version of code for targeted queries.He also points out that matrix-vector multiplication naturally corresponds to SQL join, and implementing graph mining with SQL could benefit a lot from DBMS (e.g. using optimized join alorithms).
  As for our project, we can refer to the SQL code that the paper provides and implement more queries such as *neighbors, induced subgraph, egonet,* etc.
- *Shortcomings*: Since GBASE is built on Hadoop, the paper does not mention how graph queries performs on RDBMS. Also, as the SQL version of global graph queries(*degree distribution, PageRank*, etc.) are not covered.

The second paper was the HADI paper [6].

- *Main idea*: HADI(HAdoop DIameter and radii estimator) is used for computing the radii and the diameter of Petabyte-scale graphs. To compute effecitive radius and diameter in a efficient way, HADI is designed as an approximation algorithm running on parallel platform like Hadoop, and also can be adapted to parallel RDBMS. Specially, Flajolet-Martin algorithm [Flajolet and Martin 1985; Palmer et al. 2002] is adopted for counting the number of distinct elements in a multiset for its unbiased estimation and linear time performance.
  The paper then discusses the detailed implementation and optimization of HADI on Hadoop. It also explores a little on how to adapt HADI to RDBMS system and provides a sketch of potential implementation of HADI using SQL.
  Finally, by applying HADI to mining massive real graph(YahooWeb, Twitter, etc.), the author successfully observes and reports several interesting patterns on large networks.
- *Use for our project*: One insight the paper gives is that using RDBMS for graph mining tend to be a promising direction, becuase RDBMS has a couple of advantages versus Hadoop(e.g. performance)[9].
  Also, the paper provides a solution of implementing HADI in SQL through repeated joins of edge table, as well as a unser-defined function of related operations(bit-OR in this case). This could be a good start point for us to complete the radius query task.
- *Shortcomings*: Similar to GBASE paper, HADI is fine-tuned for Hadoop platform. Therefore, for diameter/radius estimation we can explore other methods in addition to HADI which might work better on RDBMS.

The third paper was the FABP paper [8].

- *Main idea*: The paper analizes three successful methods on Guilt-by-Association: *Random Walk with Restarts(RWR), Semi-supervised Learning(SSL), and Belief Propagation(BP)*, and shows that all of them can be viewed as a matrix inversion problem.

According to the paper, Belief propagation is recommended among three methods for its solid Bayesian basis as well as capability to deal with heterophily and multiple class-lable.
The author then proposes FABP, an approxmimation method of standard BP, followed by a series of analyses and experiments indicating its advtanges in terms of performance, accuracy and scalability.

- *Use for our project*: The paper provided comprensive analyses on BP and other userful graph mining algorithms like RWR. It also gives sound proof of the FABP algorithm so that we can follow its step to implement a fast BP variant on RDBMS.
  In addition to this paper, the paper [3] also provide insights for the BP algorithm because it shares consisten implementation with the PEGASUS [7] paper.
- *Shortcomings*: Although the paper implements FABP on hadoop and show it works very well, it does not expend effort on RDBMS implementation. We need to figure out a SQL version of code based on the general description of algorithm, and see how FABP works on RDBMS.

The forth paper was the benchmark paper by A. Pavlo [9].

- *Main idea*: The paper focus on large-scale data analysis. After profound study on the architecture design of parallel DBMS and Map Reduce, the author defines and performs a comprensive benchmark test to support the earlier conclusions about the pros and cons of DBMS versus Map Reduce.
  The paper compares MapReduce frame with DBMS from 7 perspectives: *schema support, indexing, programming model, data distribution, execution strategy, flexibility, and fault tolerance*. As for the real experiment, Haddop, a column-store DBMS and a traditional row-store DBMS are used for benchmark tests. The tests includes data processing tasks(the original Map Reduce task) and analytical tasks(Data Loading, Selection, Aggregation, Join, and UDF aggregation).
  Combining theory analysis and experiment results, the paper concludes that parallel DBMS has its advantages that Map Reduce platform cannot replace, and future systems should take insights from both kinds of architectures.
- *Use for our project*: The paper bring up several insights to our project. First, it is promising to perform data mining within RDBMS. Second, column-store analytical DBMS worths our notice. Third, it shows how to rewrite Map Reduce programming paradigm with SQL using UDF. Finally, we can learn a lot from the benchmark test part when performing our analysis.
- *Shortcomings*: One fact the paper does not bring about is most of powerful parallel DBMS are commercial, while Hadoop is open-sourced and free, which contribute to an important reason why people tend to use Hadoop for large-scale data mining.

2.2 Papers read by Fangyu Gao

The first paper was the PEGASUS paper [7].

- *Main idea*: It describes why data mining in large graph is inevitable, and has to rely on MapReduce and HADOOP framework to implement large scale algorithms.
  The author unifies a bunch of algorithms in a GIM-V (Generalized Iterative Matrix-Vector multiplication) primitive, making seemly unrelated algorithms like PageRank, Random Walk, Diameter Estimation, Connected Components, etc. have a uniform abstraction.
  Using the GIM-V primitive, the author easily transferred the algorithms to the HADOOP framework. Moreover, some optimization methods can be applied on HADOOP, including Block Multiplication, which brings less number to sort and data compression, Clustered Edges, which we can reuse the preprocessing result for edge clustering for various applications, Diagonal Block Iteration, which decrease the number of iterations, Node Renumbering, which swapped minimum node id o the center node id. The author proofs that the time complexity of GIM-V is $O(\frac{V+E}{M} log \frac{V+E}{M})$, while the space complexity is $O(V + E)$.
  In the end, the author did experiments to show that all of the basic graph mining methods listed above are doing well, and some interesting discovers like power law of large scale datasets can be found thanks to the new technique.
- *Use for our project*: This paper introduces to me what graph mining can do. By searching new terminologies appears in this paper, I become to understand the goal of my whole project. Moreover, it provides me with the framework and basic methods to do graph mining in SQL.Furthermore, it equipped me with powerful optimization method I can use when facing large datasets.
- *Shortcomings*: Some trivial problems exists in the paper such as the meaning of some variables were not made clear (e.g. x in the *combineAll* from the same space).

The second paper was the HEIGN paper [4].

- *Main idea*: The eigenvalues and eigenvectors are the basic of many algorithms, but existing algorithms calculating these dont scale well on large dataset. And even billion-scale graphs are common nowadays like data from websites.
  This paper describes a method called HEigen that calculate the first several eigenvalues and eigenvectors of graph adjacency matrix and run on the MapReduce environment. Using this algorithm enabled the author to mining in large scale data, and discovered many important observations from large graph data. For example, spotting near cliques, triangle counting and eigen exponent.
  Although there are many eigensolvers commonly used when dataset is small, none of them is easily scalable to calculate more than one eigenvalue and eigenvector in very large dataset. The power method computes only the first eigenvector, QR algorithm is expensive due to using matrix-matrix multiplication, Lanczos-NO method outputs spurious eigenvalues because of rounding error.
  The author chose Lanczos-SO method that overcome all these shortcomings, and farther proposed parallelizing and a serial of optimization that make the algorithm run very fast on HADOOP.

- *Use for our project*: This paper helps me review some basic mathematical knowledge for me to complete task 5 of project 2. It also teaches me some useful optimization method to work on big data.
- *Shortcomings*: The algorithm is only applicable to symmetric matrix, so it can only calculate eigenvalues and eigenvectors of undirected-graphs.

The third paper was about Fast Couting of Triangles [10].

- *Main idea*: This paper proposed ways to fast counting of triangles in large real networks by a high accuracy way. The author used the method to count the triangles of the whole graph as well as count the triangles of a single node, which is very useful especially in social network.
  The paper introduces some non-streaming algorithms, like Brute-Force, Edge Iterator and Node Iterator, to count the number triangles in a graph exactly. Without any exception, these methods all have high computational complexity. Then, the author turns to streaming algorithms, but most of the works lack proof and justification.
  Therefore, before proposing the method, the author gives proof of the theorems he uses, including Eigen Triangle and Local Eigen Triangle. Then the proposed algorithms based on Lanczos method is given, and the author explains why the method is successful in real-world networks.
  Based on the fast and successful experimental results he gets, the author discovers new power laws, the Triangle Participation law and Degree-Triangle law.
- *Use for our project*: By learning the counting of triangles in a graph, I gain better intuitive understanding of eigenvalues and eigenvectors. The paper makes me understand why some proximate algorithms could get very high accuracy, which could help me choose the testing dataset in the project. According to the exact algorithms introduces in the paper, we could choose the most fast one for grand truth of our testing.
- *Shortcomings*: The author implements the algorithm in MATLAB. This does not mean that the algorithm will still working well in MapReduce. To proof the algorithm is robust in large scale using MapReduce, the author needs to do more experiment.

3 Proposed Method

3.1 Task List

Our goal is to implement as many tasks from the task list as possible, apply them on as many of the datasets as applicable, and study the results.

The task list includes following tasks:

- Task 1: Degree distribution
- Task 2: PageRank
- Task 3: Connected components
- Task 4: Radius and Diameter

- Task 5: Eigenvalues
- Task 6: Belief Propagation
- Task 7: Count of Triangles
- Innovation Tasks: Directed, Undirected, and Weighted Graph

3.2 Implementation Choice

We use PL/pgSQL for implementation. PL/pgSQL is the procedural language for the PostgreSQL database system. PL/pgSQL extends the functions of SQL by adding control structures(e.g. condition and loop), and thus supporting complex computations.
We consider implementing User-defined Functions(UDFs) and Stored Procedures(SPs) using SQL and register them in the DBMS server, which is a good practice for traditional OLTP to improve performance. For OLAP system and data mining tasks, though communication cost is no longer the main concern, maintaining a ready-to-use data mining package at DB server is still helpful to make data analysis easier.
Another reason for choosing PL/pgSQL that is is native to the DB system. PL/pgSQL inherits all user-defined types, functions, and operators from PostgreSQL, so that we can take advantage of the powerful feature of DBMS system itself.

3.3 Basic Operation

According to the Pegasus paper[7], most of the tasks can be represented as performing iterations on a generalized matrix-vector multiplication. Listing 1 gives the SQL code of this general implementation:

Listing 1: General Procedure

```
1  UPDATE vector
2  SET val = new_vector.val FROM
3  (SELECT matrix.row, aggregate(matrix.val*vector.val)
4    FROM matrix, vector
5    WHERE matrix.col = vector.row
6    GROUP BY matrix.row) new_vector
7  WHERE vector.row = new_vector.row
```

In addition, we find it useful to implement a set of basic operations separately for code re-usability. For example, UDF *node_lst* is used for multiple times to compute the node list given the edge list.

```
INSERT INTO node_lst
SELECT n_from AS node FROM adj_lst
UNION DISTINCT
SELECT n_to AS node FROM adj_lst ORDER BY node;
```

3.4 Task 1: Degree Distribution

3.4.1 Method Description

The major work for degree distribution task contains two steps. First, count the degree for each node(both in-degree and out-degree for directed graph). Second, for data visualization purpose(i.e. to discover power law distribution), we perform a following data processing step to count and calculate the number of nodes having the same in/out-degree.

3.4.2 SQL Implementation

We give the SQL implementation as follows:

Listing 2: Degree Distribution

```
1  CREATE degree
2    SELECT node, count(node) AS degree
3    FROM edge_lst
4    GROUP BY node;
5  SELECT degree, count(*) AS count
6  FROM degree
7  GROUP BY degree;
```

3.5 Task 2: PageRank

3.5.1 Method Description

The famous PageRank algorithm is originally invented by Google for search result ordering. The algorithm ranks web pages by adopting the rule that a web page having a lot of important web pages link to will also get a high score.
Using the power method, the algorithm can be described as iteratively multiplication on rank vector by adjacency matrix. Considering the damping points(points with no out-degree) cases, we introduce the damping factor d(by default $d = 0.85$) to enable flyout. With probability $1 - d$, random walk will fly out randomly to a new node.
The equation of PageRank can be represented as follows:

$$M = dA^T + \frac{1-d}{N}I,\ p = Mp$$

where p is the PageRank vector, A is the normalized adjacency matrix(i.e. $\sum(row) = 1$), and d is the damping factor.
Considering the scalability of the algorithm, we take advantage of the sparcity of matrix and computing with only the edges in the adjacency list.Concretely, we compute

$$p_{next} = Mp_{now} = dA^T p_{now} + \frac{1-d}{N}\mathbf{1}^T p_{now}$$

every iteration, rather than building up the full matrix and add constant to every elements at first time.
We provide algorithm 3 for a brief description.

Algorithm 1: PageRank

load data in adjacency list;
build up transposed adjacency matrix A^T given the adjacency list;
initilize PageRank vector V;
for $i \leftarrow 1$ **to** n **do**
  update p with $dA^T p_{now} + \frac{1-d}{N} \sum p_{now}$;
end

3.5.2 SQL Implementation

To compute PageRank, we first need a intialized rank vector setting every node to the uniform distribution probability(1/count of node). We implement User-Defined Function *pr_rank* to perform the job.

```
EXECUTE
'SELECT CAST(1 AS double precision)/count(*)
FROM node_lst' INTO n;

EXECUTE
'INSERT INTO pr_rank
SELECT node AS v_row, $1 AS v_val
FROM node_lst' USING n;
```

Second, we compute the normalized transposed adjacency matrix A^T using UDF *adj_matrix_trans*.

```
INSERT INTO adj_matrix
SELECT n_to AS m_row, n_from AS m_col,
CAST(1 as double precision)/degree.degree AS m_val
FROM adj_lst, degree
WHERE adj_lst_weighted.n_from = degree_weighted.node;
```

Then we update the PageRank iteratively as follows. The first EXECUTE clause compute the constant and the second EXECUTE perform the general procedure of matrix-vertex multiplication.

Listing 3: PageRank

```
1  EXECUTE 'SELECT count(*) FROM node_lst' INTO n;
2
3  FOR i IN 1..iter LOOP
```

```
    EXECUTE 'SELECT SUM(v_val) * (1 - $1) / $2 FROM pr' INTO y USING
        d, n;
    EXECUTE
     'UPDATE pr
    SET v_val = $1 * p1.v_val + $2 FROM
        (SELECT adj_matrix.m_row AS v_row, SUM(adj_matrix.m_val * pr
        .v_val) AS v_val
        FROM adj_matrix, pr
        WHERE adj_matrix.m_col = pr.v_row
        GROUP BY adj_matrix.m_row) p1
    WHERE pr.v_row = p1.v_row' USING d, y;
  END LOOP;
```

3.6 Task 3: Connected Component

3.6.1 Method Description

The main idea of Connected Component is that each node maintains and updates its connected component among the connected components from its neighbors.
To start with, we set the connected component to each node itself. For each iteration, every node receive a list of connected components from its neighbor node, then it pick the smallest node id from the list as well as its current connected component, and keep that node id as the new connected component.
The algorithm is described as follows.

Algorithm 2: Connected Component

load data into adjacency list E;
create the connected component list C;
initialize C by setting connected component to each node itself;
for $i \leftarrow 1$ **to** n **do**
  $C(i) = min(C(i), C_{(i,j) \in E}(C(j))$;
end

3.6.2 SQL Implementation

The main body of connected component algorithm is also a varient of basic operation, where the minumum function serve as the aggregate function.

Listing 4: Connected Component

```
  UPDATE conn_comp
  SET comp = n1.comp FROM
```

```
3       (SELECT adj_lst.n_from AS node, MIN(conn_comp.comp) AS comp
4       FROM adj_lst, conn_comp
5       WHERE adj_lst.n_to = conn_comp.node
6       GROUP BY adj_lst.n_from) n1
7     WHERE conn_comp.node = n1.node;
```

In addition, we also keep track of the size of Global Connected Component(GCC). In this way, we are able to stop the iteration once its value stops changing.
Here are the detailed implementation:

```
EXECUTE
'SELECT count(*) AS gcc_count
FROM conn_comp
GROUP BY comp
ORDER BY gcc_count DESC
LIMIT 1' INTO gcc_count;

IF gcc_count = gcc_last THEN
EXIT;
END IF;

EXECUTE
'INSERT INTO comp_track VALUES($1, $2)' USING i, gcc_count;

gcc_last = gcc_count;
```

3.7 Task 4: Radius

3.7.1 Method Description

In this case, we refer to HADI algorithm[6]. Concretely, we first set up 32 Flajolet-Martin bit string of length 32 for each node. Flajolet-Martin algorithm[2] approximates the number of unique objects in a stream or a database in one pass. Then during every iteration, we perform bit-wise OR among the bit-string list of the node together with the bit-string list of its neighbors.
We maintain a copy of the bit-string list to keep track of the result. As long as the bit-string for a node stops changing, we insert that node to the radius list with its radius as the number of current iteration. And we stop tracking that node by deleting it from the check list. If the check list is empty, then we finish the looping and check the radius list. All the nodes are supposed to be in the list now. And the maximum radius in the list will serve as the diameter of the whole graph.
To simplify the process, we add a self-loop for each node(i.e. input all (i, i) into edge list), which means we can perform bit-wise operation directly on the query result on neighbor

nodes from edge list.
In algorithm 5, we provide detailed description in terms of pseudo code.

Algorithm 3: Radius and Diameter

```
load data into adjacency list;
create and initialize the fm_bit string list for each node;
create and initialize the check list from bit string list;
create the radius list;
while check list is not empty do
    perform bit-or operation on the bit string list of the node and its neighbors;
    if the new bit string = the old bit string then
        add the node into the radius list;
        remove the node from the check list;
    end
    update the check list with the new bit strings;
end
diameter = max(radius list);
```

3.7.2 SQL Implementation

The first thing we do is to add self-loop via UDF *self_loop*.

```
INSERT INTO adj_lst
SELECT node as n_from, node as n_to FROM node_lst;
```

After that, we generate the 32 Flajolet-Martin bit string list for each node.

```
FOR i IN 1..n LOOP
FOR j IN 1..32 LOOP
arr[j] := 2 ^ floor(random()*32);
END LOOP;
EXECUTE 'INSERT INTO bit_str VALUES($1, $2)' using CAST(i as bigint), arr;
END LOOP;
```

Then we use the basic operation to update the bit string for each node using Bit-OR function as the aggregate funciton, which is provided by postgres here.

Listing 5: Radius

```
1  UPDATE bit_str
2  SET str = str_new.str
3  FROM(
4     SELECT adj_lst.n_from AS n_from, bit_or(bit_str.str)
5     FROM adj_lst, bit_str
```

```
6       WHERE adj_lst.n_to = bit_str.node
7       GROUP BY adj_lst.n_from) str_new
8     WHERE bit_str.node = str_new.n_from;
```

Finally, we repeat the process of removing stable node from check list and stop the iteration until the list is empty.

```
TRUNCATE temp;
INSERT INTO temp
SELECT bit_str_check.node FROM bit_str_check, bit_str
WHERE bit_str_check.node = bit_str.node AND bit_str_check.str=bit_str.str;

EXECUTE 'INSERT INTO node_radius (SELECT node, $1 FROM temp)' using round;

DELETE FROM bit_str_check WHERE node IN
(SELECT node FROM temp);

SELECT COUNT(*) FROM bit_str_check into count;
IF count = 0 THEN
EXIT;
END IF;

UPDATE bit_str_check
SET str=bit_str.str
FROM bit_str
WHERE bit_str_check.node = bit_str.node;
```

3.8 Task 5: Eigenvalues

3.8.1 Method Description

We implemented the Lanczos-NO algorithm in the paper HEigen: Spectral Analysis for Billion-Scale Graphs[4], the algorithm is described as follows.

3.8.2 SQL Implementation

Most of the operations in this task are matrix multiplications and vector multiplication. The implement of these are the same as before.

Listing 6: Eigenvalues

```
1 CREATE OR REPLACE FUNCTION task_5(a_matrix text, b text, m integer
      , k integer)
2 RETURNS void AS
```

Algorithm 4: Eigenvalues

generate a vector with size equal to the column number of adjacency matrix; let it be the initial base v_i;
for $i \leftarrow 1$ **to** n **do**
    Find a new basis vector;
    Orthogonalize against two previous basis vectors;
    build tri-diagonal matrix;
    Eigen decomposition of the tri-diagonal matrix;
    selectively re-orthogonalize the new basis vector;
end
build the last tri-diagonal matrix;
Eigen decompose the last tri-diagonal matrix;
compute the top k diagonal elements;
compute eigenvectors;

```
$_$
DECLARE
  betai_1 double precision := 0;
  betai double precision;
  i integer;
  alphai double precision;
BEGIN
  EXECUTE create_b(b, a_matrix);
  EXECUTE create_vector('vi_1');
  EXECUTE create_matrix('v_matrix');
  EXECUTE create_vector('alpha');
  EXECUTE create_vector('beta');
  EXECUTE vector_scale('vi', b, 1/vector_length(b));
  FOR i IN 1..m LOOP
    EXECUTE assign_matrix_column('v_matrix', i, 'vi');
    EXECUTE matrix_multiply_vector('v', a_matrix, 'vi');
    alphai := vector_multiply_vector('vi', 'v');
    EXECUTE assign_vector_value('alpha', i, alphai);
    EXECUTE vector_scale('temp', 'vi_1', betai_1);
    EXECUTE vector_minus('v', 'v', 'temp');
    EXECUTE vector_scale('temp', 'vi', alphai);
    EXECUTE vector_minus('v', 'v', 'temp');
    betai := vector_length('v');
    IF i != m
    THEN
      EXECUTE assign_vector_value('beta', i, betai);
```

```
29      END IF;
30      EXIT WHEN betai = 0;
31      EXECUTE assign_vector('vi_1', 'vi');
32      EXECUTE vector_scale('vi', 'v', 1/betai);
33      betai_1 := betai;
34    END LOOP;
35    EXECUTE create_tridiagonal_matrix('tm', 'alpha', 'beta');
36    EXECUTE eig('q', 'd', 'tm');
37    EXECUTE eigenvalue('lambda', 'd', k);
38    EXECUTE matrix_corp_column('qk', 'q', 1, k);
39    EXECUTE matrix_multiply_matrix('rk', 'v_matrix', 'qk');
40  END;
41  $_$ LANGUAGE plpgsql;
```

3.9 Task 6: Belief Propagation

3.9.1 Method Description

Belief Propagation is a commonly-used message passing algorithm to perform inference on graphical model, such as Bayesian networks(directed graph) and Markov random fields(undirected graph).
As for algorithm, we implemented FastBP[8], a fast approximate BP algorithm guaranteed to converge. The equation we refer to are as follows.

$$M = aDcA,\ b^{t+1} = Mb^t,\ belief = \sum(b)$$

$$D = \sum_j A_{ij}\ and\ D_{ij} = 0\ for\ i \neq j$$

$$c_1 = 2 + \sum_i d_{ii},\ c_2 = \sum_i {d_{ii}}^2 - 1,\ h_h < \sqrt{\frac{-c_1 + \sqrt{c_1^2 + 4c_2}}{8c_2}}$$

$$a = \frac{4h_h^2}{1 - 4h_h^2},\ c^{'} = \frac{2h_h}{1 - 4h_h^2}$$

The implementation of Belief Propagation algorithm is similar to PageRank. We sum over the result of vector after each iteration to get the final approximate of belief for each node. Also, we ultilize the sparse matrix. Instead of generating the whole matrix, we only compute the elements in the edge list as well as on the diagonal, which means great news for large-scale graph mining.
The algorithm of FastBP is described as follows.

Algorithm 5: Belief Propagation

load data into the adjacency list, which also serves as adjacency matrix Ahere;
load data into the initial belief list b;
create and compute trace matrix D from A;
compute c_1, c_2, and h_h using D;
compute a and $c^{'}$ using h_h ;
scale A and D using a and $-c^{'}$; union the result as the matrix M;
create final belief list b_{final}; initial it using b;
while b_{final} *does not converge* **do**
    update b with Mb;
    update b_{final} by adding b;
end

3.9.2 SQL Implementation

Before starting the inference process, we first compute the related parameters to be used.

```
CREATE TABLE d_matrix AS
SELECT m_row, m_row as m_col, SUM(m_val) as m_val from adj_matrix GROUP by m_row;

EXECUTE 'SELECT SUM(m_val) + 2 from d_matrix' INTO c1;
EXECUTE 'SELECT SUM(m_val^2) - 1 from d_matrix' INTO c2;

hh = sqrt((-c1 + sqrt(c1 ^ 2 + 4 * c2)) / (8 * c2));
a = 4 * (hh ^ 2) / (1 - 4 * (hh ^ 2));
c_prime = 2 * hh / (1 - 4 * (hh ^ 2));

EXECUTE
'CREATE TABLE w_matrix AS
(SELECT m_row, m_col, $1 * m_val AS m_val FROM adj_matrix)
UNION
(SELECT m_row, m_col, $2 * m_val AS m_val FROM d_matrix)' USING a, (-c_prime);
```

Then we create two vectors and initiazed them with the proir belief. After that, we kick off the iteration preocess. The main body of iteration follows the basic matrix-vector multiplication paradigm.

Listing 7: Belief Propagation

```
1  UPDATE b_vector
2  SET v_val = b_new.v_val
3  FROM(
4     SELECT m_row AS v_row,
5     SUM(m_val * b_vector.v_val) AS v_val
```

```
    FROM w_matrix, b_vector
    -- FROM adj_matrix, b_vector
    WHERE m_col = b_vector.v_row
    GROUP BY m_row) b_new
WHERE b_vector.v_row = b_new.v_row;

UPDATE belief
SET v_val = belief.v_val + b_vector.v_val FROM b_vector
WHERE belief.v_row = b_vector.v_row;
```

3.10 Task 7: Count of Triangles

3.10.1 Method Description

We implemented the EigenTriangle function in the paper Fast Counting of Triangles in Large Real Networks: Algorithms and Law[10].
The paper proofs: The total number of triangles in a graph is proportional to the sum of cubes of eigenvalues.

$$\Delta(G) = \frac{1}{6}\sum_{i=1}^{n}\lambda^3$$

3.10.2 SQL Implementation

After getting the eigenvalue and eigenvector from task 5, the implementation of this task is just a few line of sql code.

Listing 8: Count of Triangle

```
select matrix_row, sum((matrix_value^2) * (vector_value^3))/2
from matrix, vector
where matrix_column = vector_row
group by matrix_row;
```

3.11 Innovation Tasks: Directed, Undirected, and Weighted Graph

As the default graph type is undirected graph, what we are trying to do for innovation is to extend the graph type to directed and weighted graph.
Specifically, we extend task 1, 2, 3, 4, 6 to directed graph, and we also implement task 1, 2, 5 for weighted graph.
Here we will briefly discuss the methods and thoughts based on each task. And we will use them to analysize graphs in different types in the experiment section.
One common UDF commonly used is *undirected_proc*, since for some task we need to tranform a directed graph into undirected first. Actually the UDF also works for those uni-

directional undirected graphs.
Here we give the SQL of *undirected_proc* as follows.

Listing 9: Undirected Procedure

```
CREATE TABLE undirected_proc AS
  SELECT n_from, n_to FROM adj_lst
  UNION DISTINCT
  SELECT n_to, n_from FROM adj_lst;

DROP TABLE adj_lst;
ALTER TABLE undirected_proc RENAME TO adj_lst;
```

- *Degree*: As for directed graph, we compute in-degree and out-degree seperately.
  As for weighted graph, we compute weight together with degree. That means for weigted directed graph, we compute in-weight and out-weight.
- *PageRank*: PageRank naturally works on directed graph and no change has to make here.
  As for weighted graph, the change happens in the adjacency matrix normalization. In stead of using degree as measurement, here we consider the weight contribution.
- *Connected Component*: There are two types of connected component for directed graph. What we implement here is weakly connected component, which extend directed graph to bi-directional. Therefore, our steps are first applying UDF *undirected_proc*, and then use the connected component method for undirected graph to compute weakly connected component for directed graph.
- *Radius*: The situation is similar to the former task. Radius method works directly on directed graph. As it does not make too much sense in data mining to consider the weight as distance, there is no change for weighted graph either.
- *Belief Propagation* As for directed graph, belief propagtion works direcly.

4 Experiments

4.1 Dataset

Table 2 gives the current datasets we used for test. Datasets come from SNAP and KONECT.
web-Google dataset is Web graph from Google.
social-Youtube dataset is Youtube online social network.
roadMap-PA(CA/TX) dataset is Road network of Pennsylvania(California/Texas).
rating-Libimseti dataset is the network of ratings given by users of Libimseti.cz to other users. The network is unipartite, directed, and edges represent ratings on a scale from 1 to 10. p5cm

Besides that, we also define four typical test cases to indicate algorithm accuracy.
Each test case has 5 nodes, as show in Figure 1.

Dataset	Type	Nodes	Edges
web-Google	Directed	875,713	5,105,039
social-Youtube	Undirected	1,134,890	2,987,624
roadMap-PA	Undirected	1,088,092	3,083,796
roadMap-CA	Undirected	1,965,206	5,533,214
roadMap-TX	Undirected	1,379,917	3,843,320
rating-Libimseti	Directed/Weighted	220,970	17,359,346

Table 2: Test Datasets

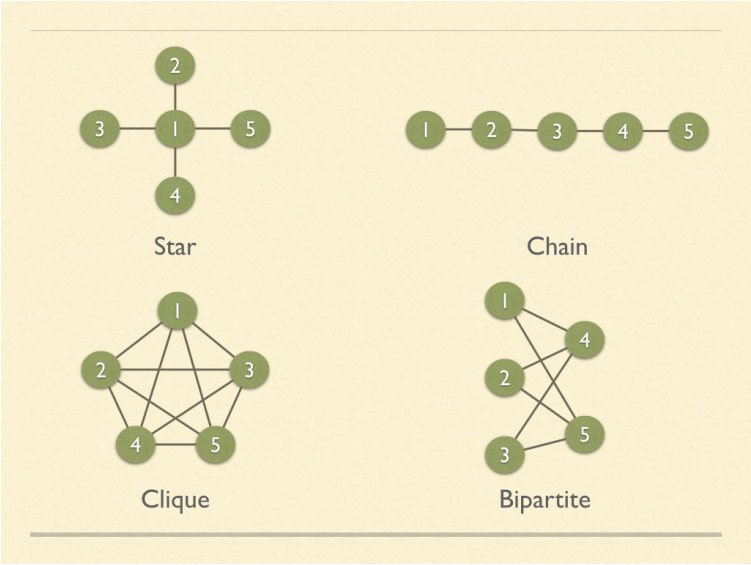


Figure 1: Test Case

4.2 Task 1: Degree Distribution

4.2.1 Accuracy Demonstration

One way we use the veryify the result is to use the task case(which can be inferred manually). The reults we get from algorithm are as follows, which meet the values we expected.

```
Star:
 degree | count
--------+-------
      1 |     4
      4 |     1

Chain:
 degree | count
```

Case	Description
Star	one node serves as a hub, the rest of nodes form 1 degreee connect with hub and not connected to each other.
Chain	one node connected to a precedent and a follower node
Clique	every node has full connection to the rest of nodes in the group
Bi-partite	nodes are divided into two group, node in one group has full connection to the nodes at the other group, but no connection to the nodes at the same group

Table 3: Test Cases

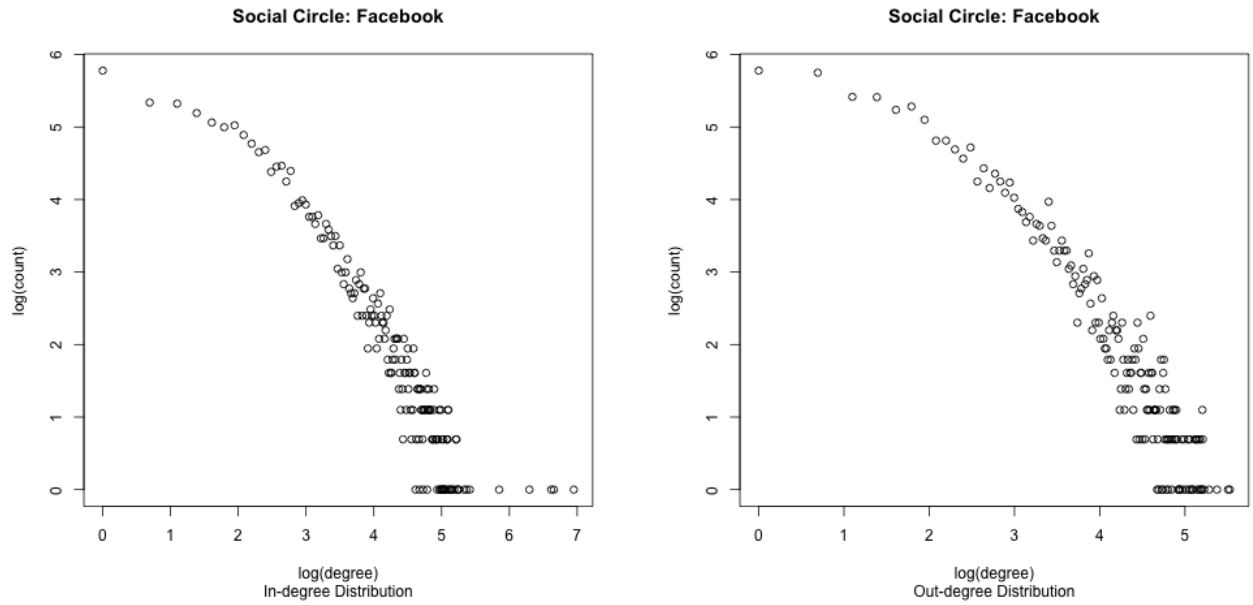
```
--------+-------
      1 |     2
      2 |     3

Clique:
 degree | count
--------+-------
      4 |     5

Bipartite:
 degree | count
--------+-------
      2 |     3
      3 |     2
```

Another way is to check if the plot follows the power law in real dataset. We implemented the degree distribution method using SQL and performed the query on Facebook Social Circle dataset from SNAP [1]. The graph has 4039 vertices and 88234 edges.

Figure 2 shows our results: Figure 2(a) gives a scatter-plot of the in-degree distribution on a log-log scale for the Facebook graph dataset. Figure 2(b) shows the out-degree distirbution.

[1]SNAP: http://snap.stanford.edu/data/index.html

X axis indicates the in-degree and y axis indicates the number of vertices which have the same in-degree or out-degree. According to the figures, We can easily observe the Power Law(heavy-tailed degree distribution) pattern.

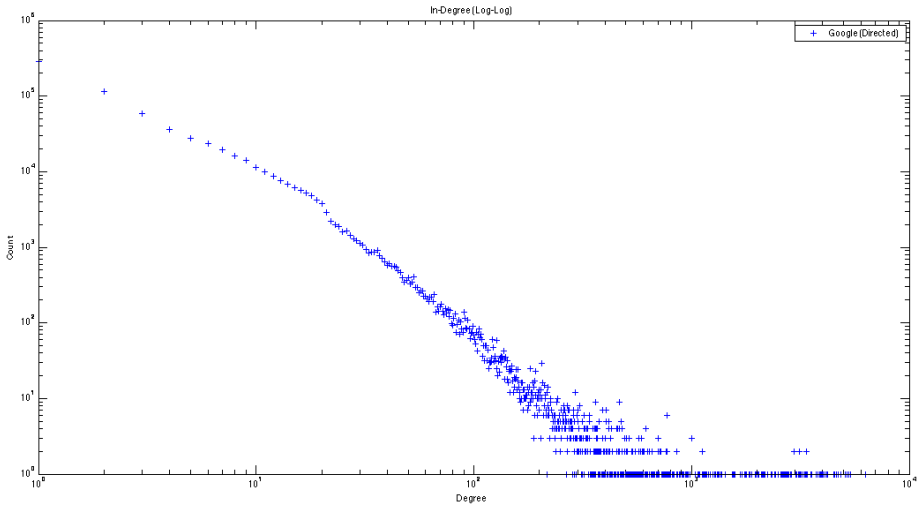


(a) (b)

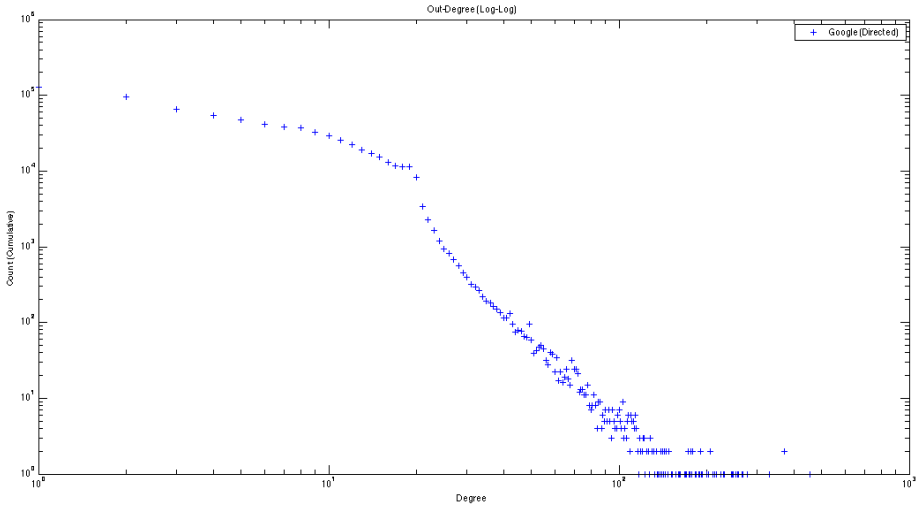
Figure 2: The in-degree distribution plot(a) and out-degree distribution plot (b) on a log-log scale for Facebook graph dataset

4.2.2 Variety Demonstration

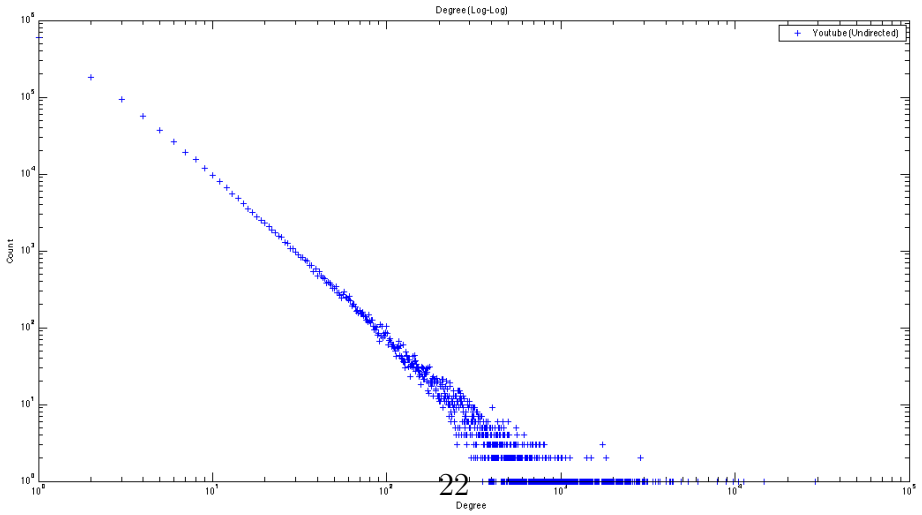
We perform the test on five 1M node datasets: web-Google(directed), social-Youtube, roadMap-PA, roadMap-CA, and roadMap-TX. Figure 4 shows the result.
From the result we can find the heavy-tailed power law distribution from every figure, which shows that real dataset are very skewed distributed.
What we can also find is the difference between social network/web graph and road network graph. Road network turns out to have very small number in terms of degree. This actually make sense because degree represent the number of direct neighbours. While a webpage can link to hundreds of other webpages, or a person could have millions of follower on social network, a road in real life actually connect to only a few of roads.



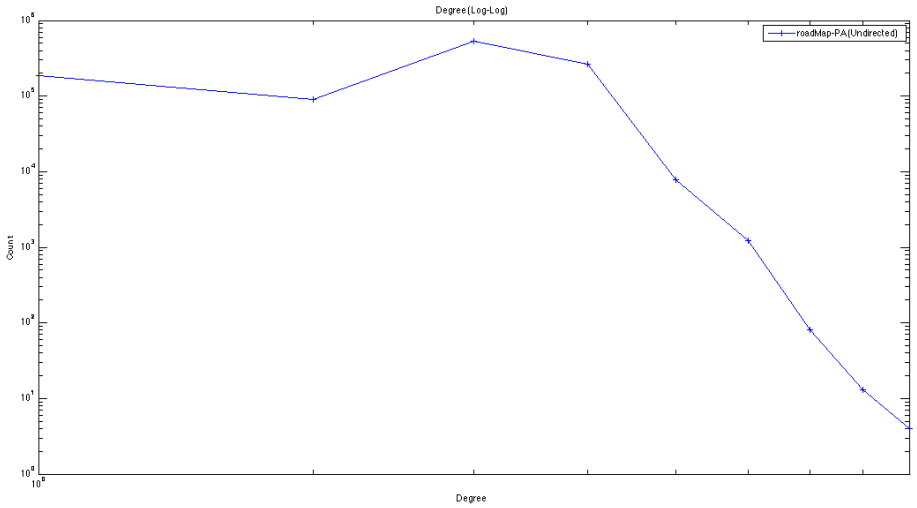
(a) Google in-Degree



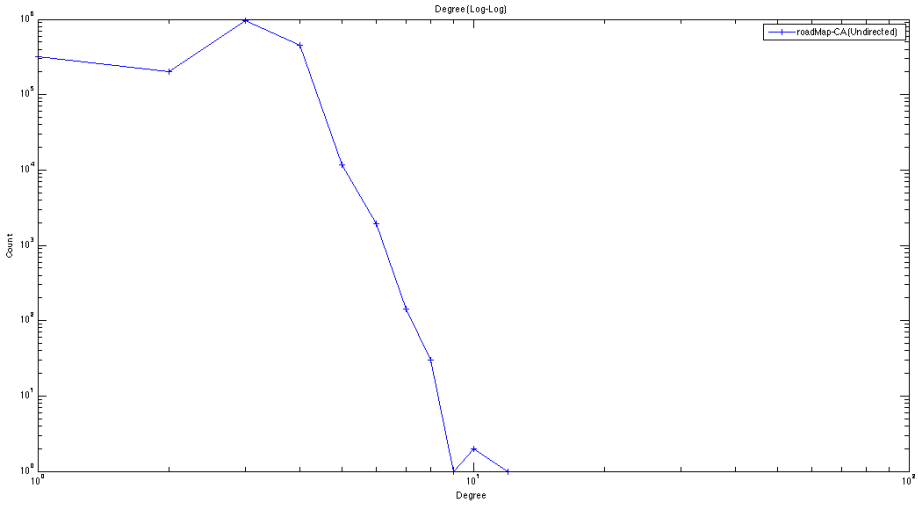
(b) Google out-Degree



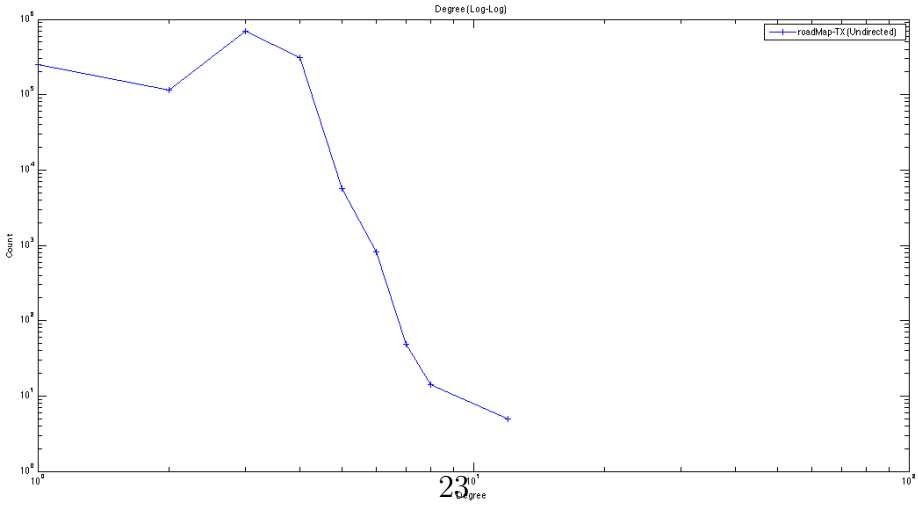
(c) Youtube



(a) roadMap-PA



(b) roadMap-CA



(c) roadMap-TX

Figure 4: Degree Distribution

4.3 Task 2: Page Rank

4.3.1 Accuracy Demonstration

We can also use the test case to tell if the PageRank computed by the algorithm is accurate. Consider the directed version of Star test case, where the hub(node 1) is pointed from every other nodes(node 2,3,4,5).
The result we get actually reflect the graph structure, where the hub has a higher rank and the rest of nodes share the same lower rank.

```
 v_row |       v_val
-------+--------------------
     1 |                0.2
     2 | 0.0551136364533462
     3 | 0.0551136364533462
     4 | 0.0551136364533462
     5 | 0.0551136364533462
```

Another thing we need to verify is the convergence of PageRank. Here we give the number of rank in iteration 10 and iteration 100. The result indicates that the rank computed by our algorithm coverges.

```
Iteration 10:
 v_row |       v_val
-------+--------------------
     1 |                0.2
     2 | 0.0551136364533462
     3 | 0.0551136364533462
     4 | 0.0551136364533462
     5 | 0.0551136364533462
Iteration 100:
 v_row |       v_val
-------+--------------------
     1 |                0.2
     2 | 0.0551136363636364
     3 | 0.0551136363636364
     4 | 0.0551136363636364
     5 | 0.0551136363636364
```

4.3.2 Variety Demonstration

We perform the test on five 1M node datasets: web-Google(directed), social-Youtube, roadMap-PA, roadMap-CA, and roadMap-TX. Figure 6 shows the result.
Notice that we are using cumulative count here in terms of the y-axis. That is, while x is the PageRank of a specific point, y is the number of points with greater PageRank value than

that point.
From the figure we can figure out that both Youtube and Google graph follow the Power Law distribution of PageRank, which is pointed out in the paper [1], while the roadMap datasets behave a little differently: only the latter part of the plot looks like power low distribution.

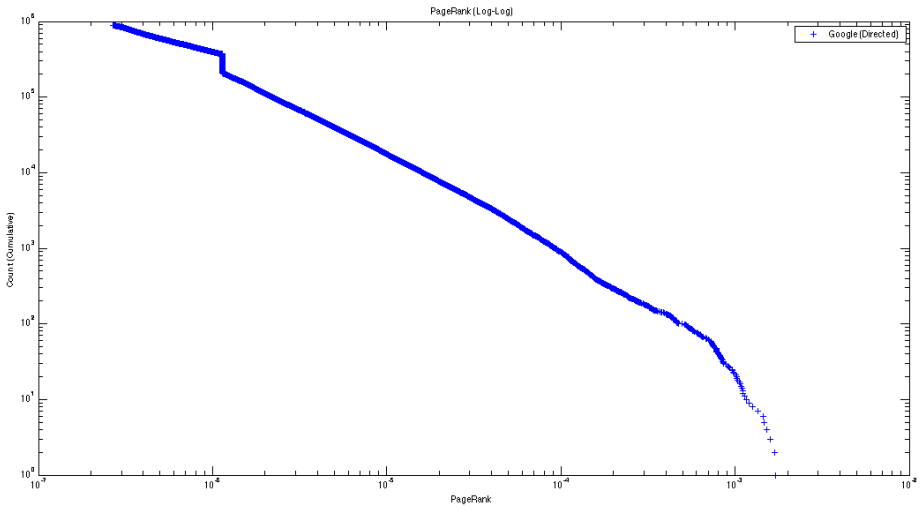
4.4 Task 3: Connected Components

4.4.1 Accuracy Demonstration

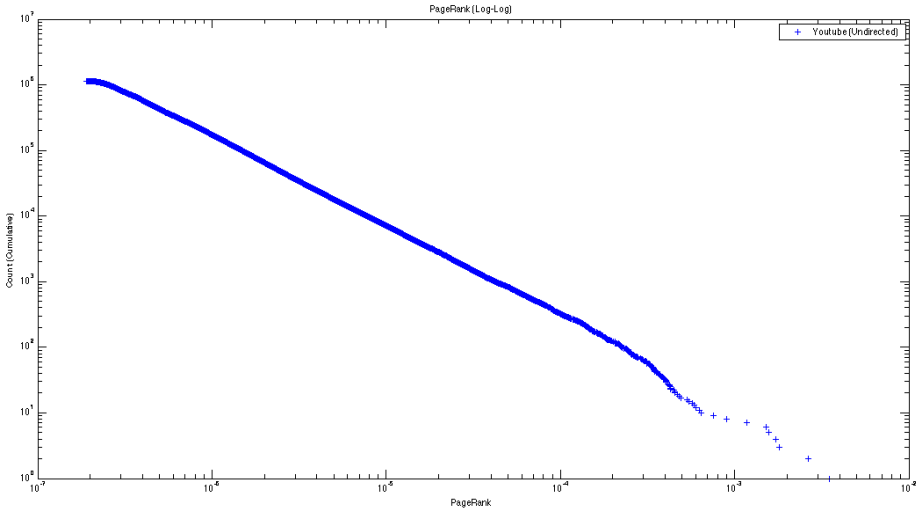
We use the test case to verify if the connected components computed by the algorithm is accurate.
Here we combines 4 test cases into one graph: node 1-5 belongs to Star graph; node 11-15 belongs to chain graph; node 21-25 belongs to clique graph; node 31-35 belongs to bipartite. As for result, we expect to see four component each with size of 5. And the result are given as follows, which satisfies our expectation.

node	comp
1	1
2	1
3	1
4	1
5	1
11	11
12	11
13	11
14	11
15	11
21	21
22	21
23	21
24	21
25	21
31	31
32	31
33	31
34	31
35	31

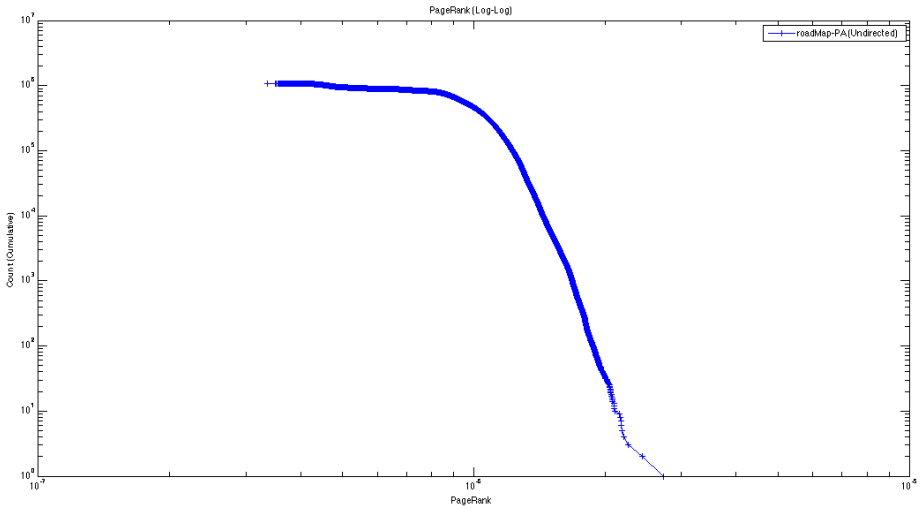
As for the convergence test, we modify the test case above by connecting four sub graph together, i.e. add three edges: 1-11, 11-21, 21-31. From the result given, we can tell that it costs five round for the algorithm to converge and the whole graph is formed in to a giant connected component, which is exactly what we expect.



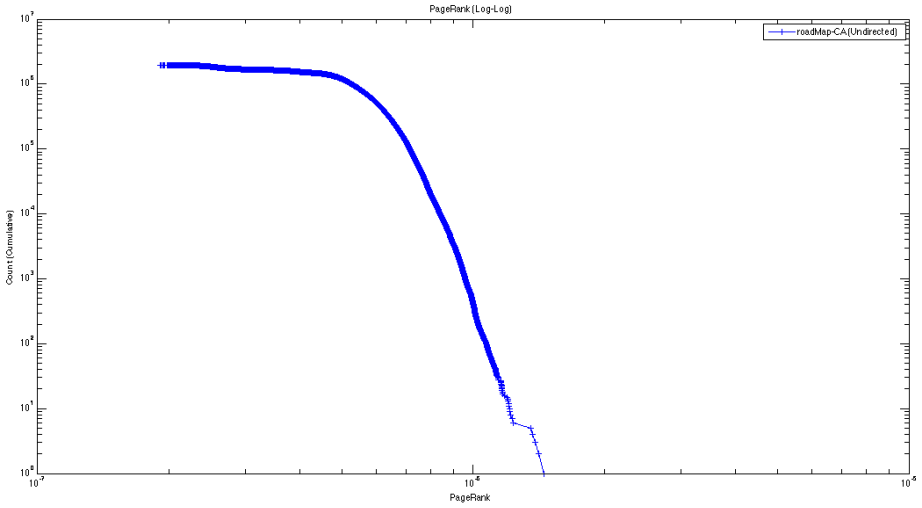
(a) Google



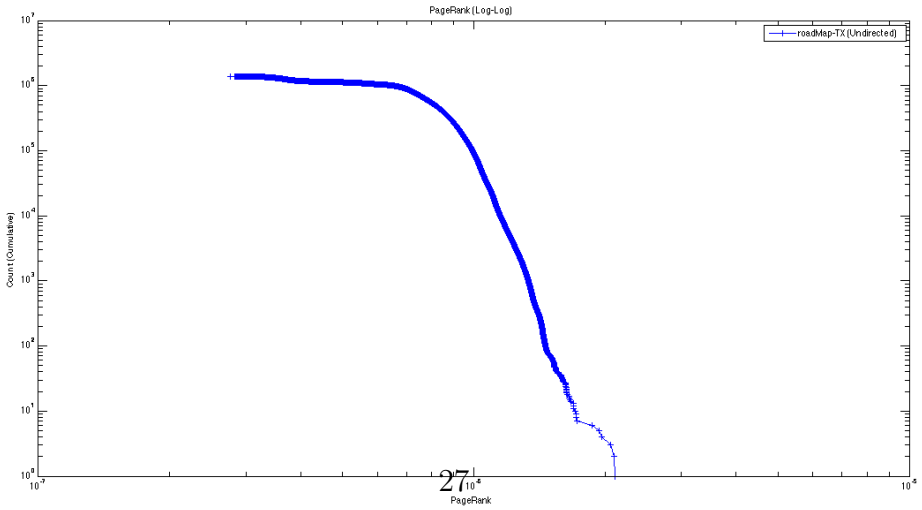
(b) Youtube



(a) roadMap-PA



(b) roadMap-CA



(c) roadMap-TX

Figure 6: PageRank Distribution

```
 node | comp
------+------
    1 |    1
    2 |    1
    3 |    1
    4 |    1
    5 |    1
   11 |    1
   12 |    1
   13 |    1
   14 |    1
   15 |    1
   21 |    1
   22 |    1
   23 |    1
   24 |    1
   25 |    1
   31 |    1
   32 |    1
   33 |    1
   34 |    1
   35 |    1
(20 rows)

Time: 0.813 ms
 round | gcc_count
-------+-----------
     1 |         6
     2 |         8
     3 |        14
     4 |        17
     5 |        20
```

4.4.2 Variety Demonstration

We perform the test on five 1M node datasets: web-Google(directed), social-Youtube, roadMap-PA, roadMap-CA, and roadMap-TX. Figure 7 shows the result of convergence.
From the figure, we can see that Google and Youtube graph converge in around 10 iterations. However, roadMap-PA graph did not reach to convergen after the maximum iteration(which is set to 256).
The big difference from social network/web graph and roadnet graph still exists here. Although it takes only a few iteration for the network/graph to converge, the road graphs take far more rounds. The situation makes good sense because it exactly reflect the effect of small

diameter/small world/six degree of seperation in terms of social network and web graph. Meanwhile, the common sense confirms us that road networks are supposed to have long chain and large radius/diameter, which means far more iterations for the connected components to converge.

We also give the plot of connected component distribution. Notice that Youtube graph is actually all-connected and there is only one GCC. Therefore, we only give the plot of Google, roadMap-PA after 20 iterations, roadMap-PA after 256 iterations(which took over 22000000 ms to run), roadMap-CA after 10 iterations and roadMap-TX after 10 iterations. Figure 11 shows the result of component plot.

4.5 Task 4: Radius

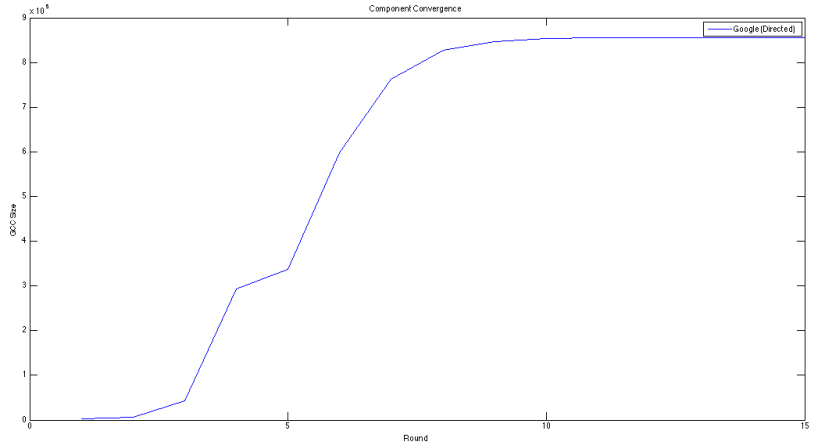
4.5.1 Accuracy Demonstration

We examine a small test case on a undirected graph with 8 nodes. Figure 10 shows the graph structire.
It is obvious to tell from the image that Node 1 and Node 5 have the largest radium 5, which is the diameter of this graph.
Running the algrithm on the graph, we get the output as follows, which is in line with the result from the image.

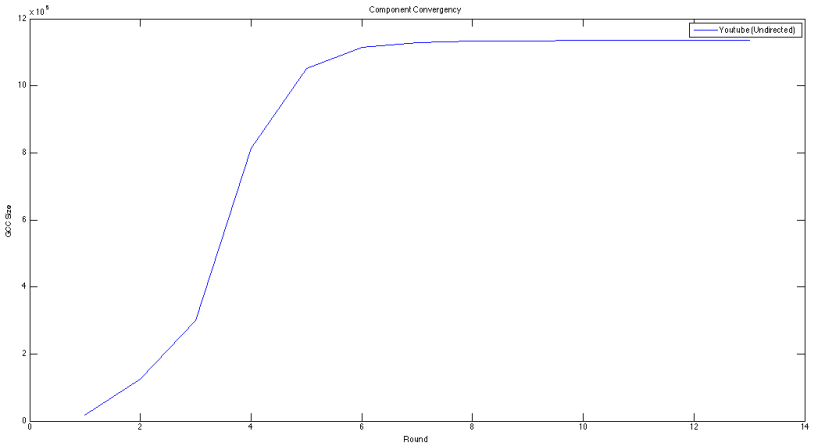
```
postgres=# select * from radius(20);
 node | radius
------+--------
    1 |      5
    2 |      4
    3 |      3
    4 |      4
    5 |      5
    6 |      2
    7 |      2
    8 |      2
(8 rows)
```

4.5.2 Variety Demonstration

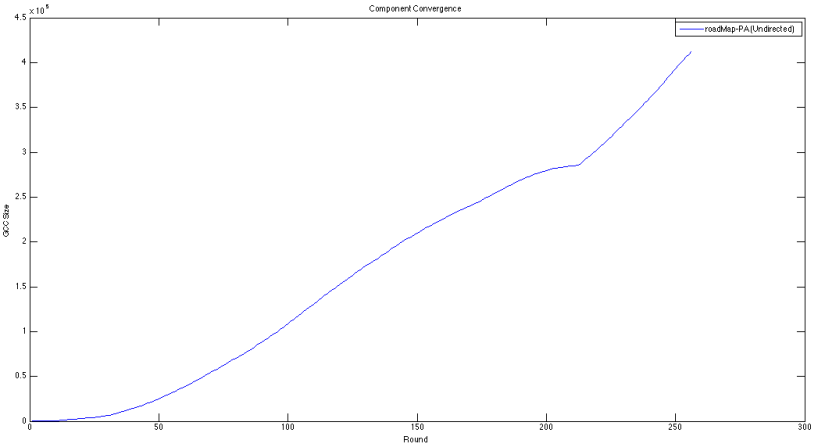
We perform the test on two 1M node datasets: web-Google(directed) and Youtube. Figure 11 shows the result.
Since the iteration stop earlier than maximum iteration(256), we guarantee that radius are converge here.
As we mentioned in the last section that real world graphs like road network tend to have chains and large radius, we skip the test here considering its running time.



(a) Google

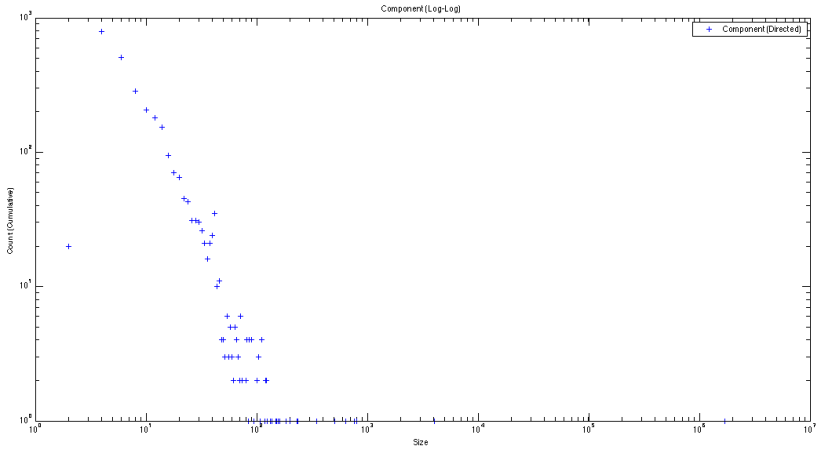


(b) Youtube

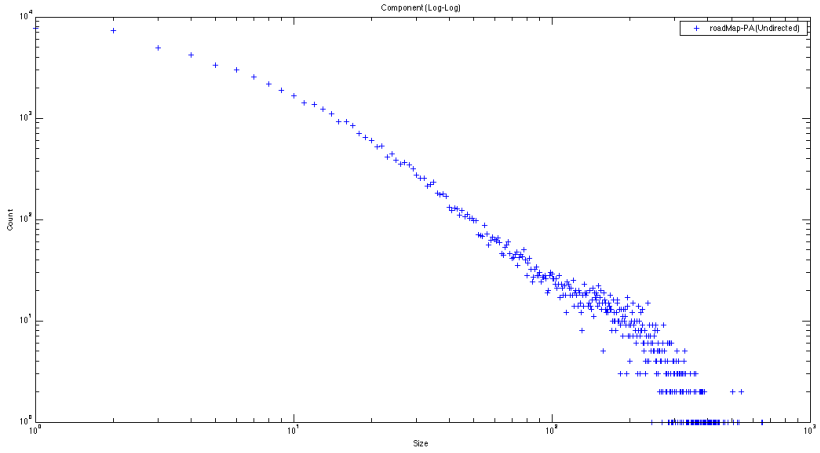


(c)

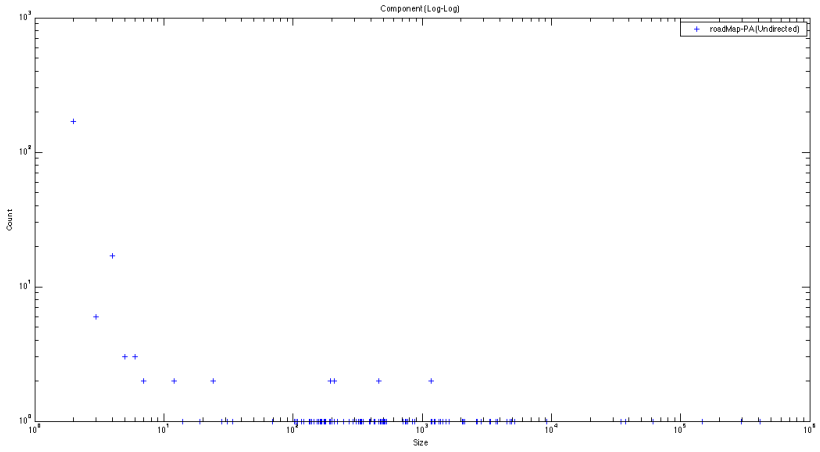
Figure 7: Component Convergence



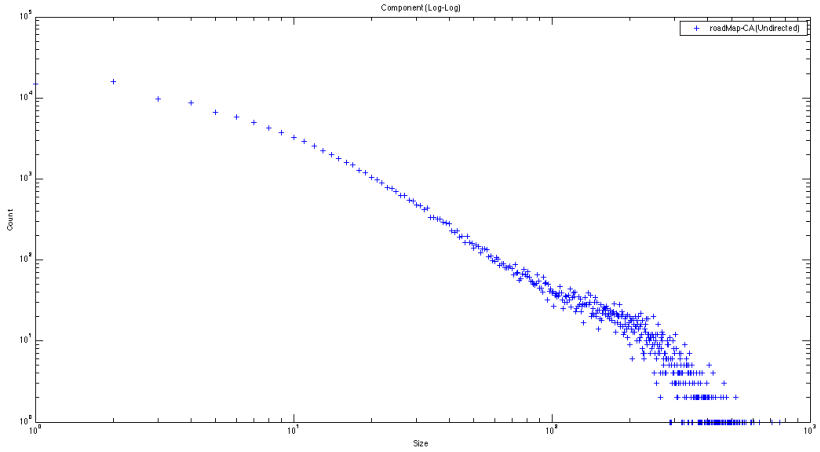
(a) Google



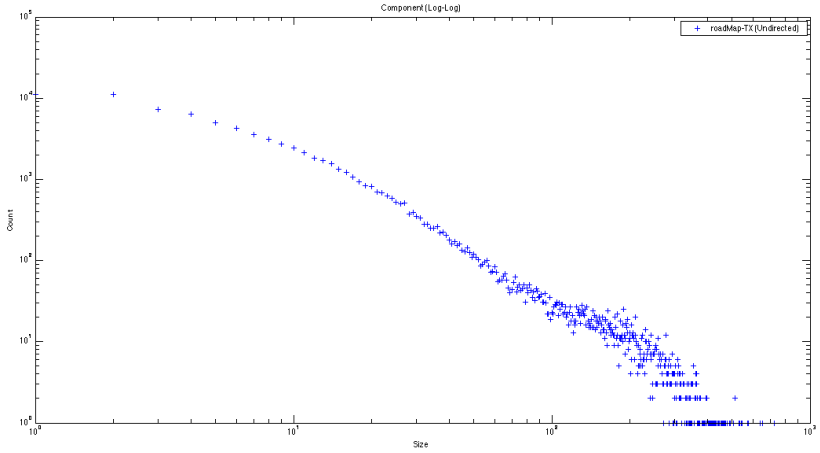
(b) roadMap-PA 20 iterations



(c) roadMap-PA 256 iterations



(a) roadMap-CA 10 iterations



(b) roadMap-TX 10 iterations

Figure 9: Component Distribution

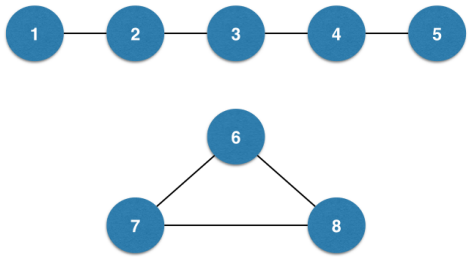
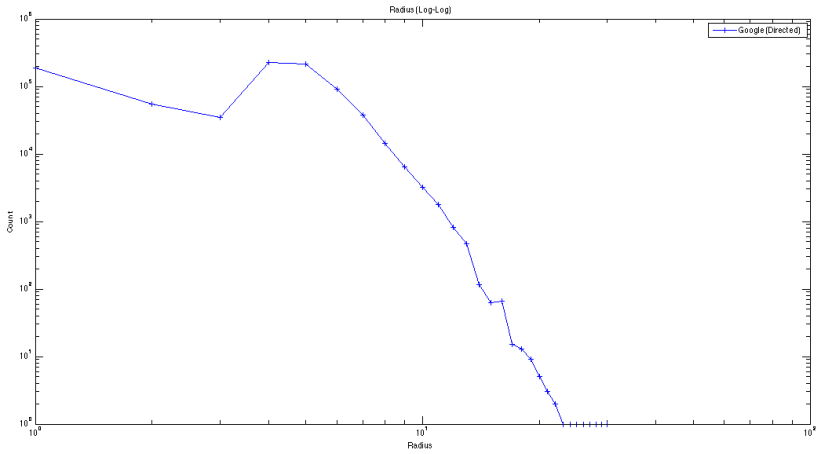


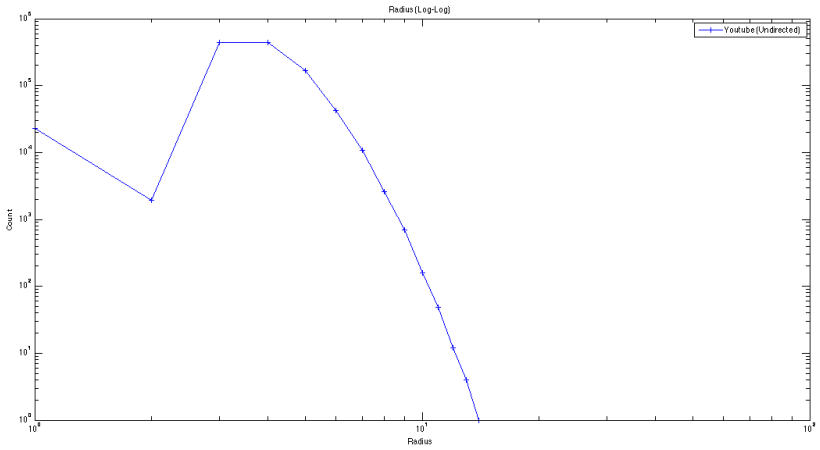
Figure 10: Test case graph for Radium and Diameter

4.6 Task 5: Eigenvalues

Accuracy Demonstration First, we successfully implemented Lanczos-NO algorithm. But we found there were some eigenvalues missing. We thought it was because the rounding errors from floating-point calculations as its said in the paper. But after we implemented Lanczos-SO algorithm, there are still eigenvalues missing.
In the following test, we found out that the correctness of calculating eigenvalues by Lanczos is largely related by the number of iteration and the initial value chosen. The more the number of iteration, the more correctness brought to eigenvalue calculation, but the also the more calculation time it takes.
As a treat off, we only calculate the first 10 eigenvalues in the following experiments, as shown in Figure 12.
As can be seen from the eigenvalues calculated by us and the ones calculated by MATLAB. Only the first few result are corresponding to the greatest eigenvalue that calculated by MATLAB.
Our conclusion is that Lanczos algorithm with small number of iteration K, is not suitable to calculate the greatest K eigenvalues of a matrix, because it may not find them all, like the youtube dataset. And the calculation result is extremely bad when eigenvalues are close to each other, like roadnet datasets.
Despite of this, we found out some pattern of the eigenvalues calculated by the Lanczos algorithm. Take a simple dataset - adjnoun - as example. The eigenvalue-rank plot is as follows in Figure 13.
As can be seen in the graph, the first few eigenvalues are alternately greater or smaller than 0, and the absolute value of eigenvalues are getting smaller.



(a) Google



(b) Youtube

Figure 11: Radius Distribution

youtube		roadnet_PA		roadnet_CA		dataset
PostgreSQL	MATLAB	PostgreSQL	MATLAB	PostgreSQL	MATLAB	method
210.3952	210.3953	3.998	4.4195	4.239	4.6384	eigen1
-176.5156	-176.5188	3.5962	4.2887	3.6889	4.527	eigen2
154.4618	169.4289	-3.2711	4.2635	-3.2813	4.4516	eigen3
-133.9068	154.8155	3.1576	4.2351	3.2205	4.3903	eigen4
-108.0994	-138.5002	-2.6369	4.2292	-2.6171	4.3837	eigen5
104.7654	137.6521	2.4546	4.1994	2.5667	4.3257	eigen6
59.7785	-128.0105	-1.7653	4.1982	-1.7339	4.2866	eigen7
-50.9715	109.5985	1.4355	4.1709	1.5447	4.2785	eigen8
13.6972	-108.6547	-0.6684	4.1315	-0.6287	4.2641	eigen9
-2.7184	108.038	0.2573	4.1053	0.326	4.2617	eigen10

Figure 12: Eigenvalues and rank plot

4.6.1 Varienty Demonstration

We perform the test on several 1M nodes datasets: social-Youtube, roadMap-PA and roadMap-CA. Figure 14 shows the result.

4.7 Task 6: Belief Propagation

4.7.1 Accuracy Demonstration

First, We want to check if the algorithm is able to give the correct inference of unlabed node. We examine a small test case on a undirected graph with 4 nodes. Figure 15 shows the graph structure. Given the label and initial belief of node 1, 2, 3, we intend to infer the label of node 4.
As for the intial belief, we set node 1=0.4(label: +), node 2=-0.4(label: -), node 3=0.3(label: +), and we leave node 4 = 0(lable: unknown). From the architecture of the graph, we can easily reach the conclusion that node 4 is more likely to be labeled as +(i.e. belief(node 4) ¿ 0).
After performing the FastBP algorithm on the test graph, we have the output as follows. The node 4 is classified as +, which is what we expect.

```
postgres=# SELECT * from bp(20);
 v_row |       v_val
-------+--------------------
     1 |  0.273209372007041
     2 | -0.198949408545921
```

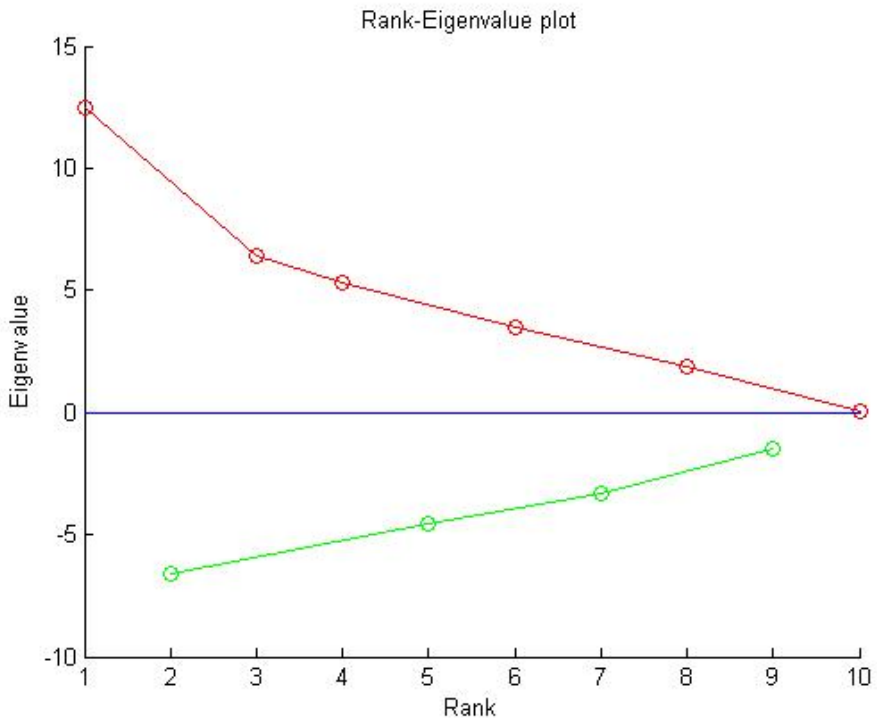
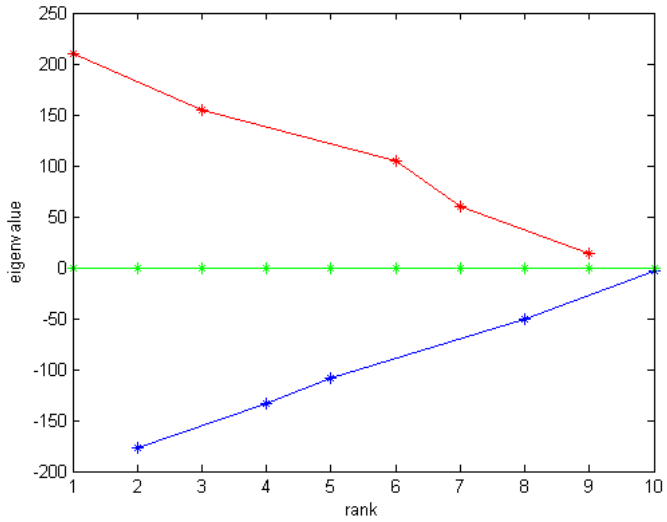


Figure 13: Eigenvalues-rank plot for adjnoun dataset

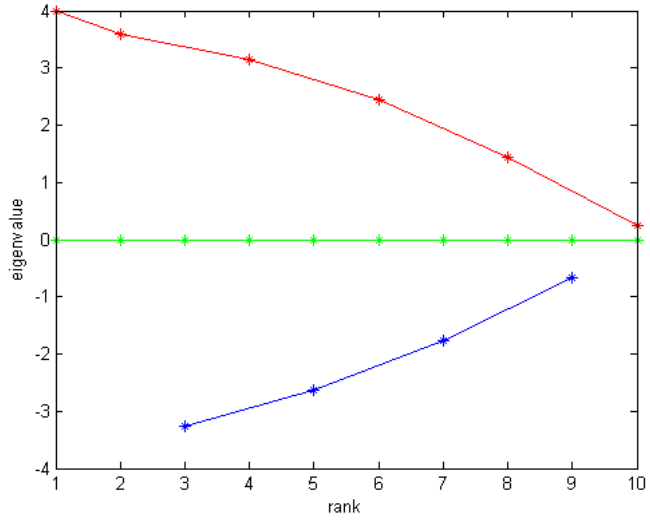
```
     3 |  0.159512788522935
     4 | 0.0141145717811318
(4 rows)
```

One thing worths noticing is that as h_h serve as about-half homophily factor according to FastBP paper[8]. h_h close to -0.5 means strong heterophily, while h_h close to 0.5 means strong homophily. Therefore, we can also heterophily inference on the graph by setting h_h to be negative.
Second, we want to check if the algorithm can converge. We compare the result after 10 iterations, 50 iterations and 100 iterations, which shows that the FastBP algorithm does converge.

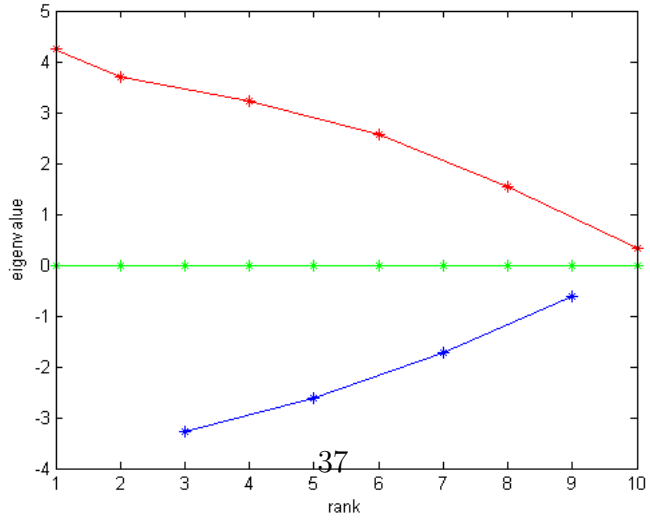
```
10 iterations:
 v_row |         v_val
-------+-----------------------
     1 |     0.288775545358658
     2 |    -0.264187325537205
     3 |     0.222905980050564
     4 | -0.000686401128768922
```



(a) Youtube



(b) roadMap-PA



(c) roadMap-CA

Figure 14: Eigenvalues and rank plot

```
50 iterations:
 v_row |       v_val
-------+--------------------
     1 |  0.273209372007041
     2 | -0.198949408545921
     3 |  0.159512788522935
     4 | 0.0141145717811318

100 iterations:
 v_row |       v_val
-------+--------------------
     1 |  0.272932111246243
     2 | -0.197774913230647
     3 |  0.158338293512337
     4 | 0.0143918324157287
```

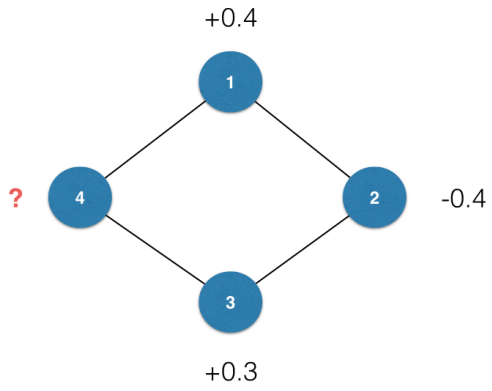


Figure 15: Test case graph for Belief Propagation

4.7.2 Varienty Demonstration

We perform the test on five 1M node datasets: web-Google(directed), social-Youtube, roadMap-PA, roadMap-CA, and roadMap-TX. As for initialization, we set the prior belief to 0.01 for all nodes.
Figure 17 shows the result.

Notice that we are using cumulative count here in terms of the y-axis. That is, while x is the belief of a specific point, y is the number of points with greater belief value than that point.

4.8 Task 7: Count of Triangles

4.8.1 Accuracy Demonstration

As we talked about in task 5, the eigenvalues calculated by Lanczos-SO are not all the eigenvalues of the adjacency matrix nor are the greatest values. As we do the experiment, the triangle counting are not accurate because of this.
The experiment result is as follow in Figure 18.
As can be seen by comparing the result of PostgreSQL implementation and MATLAB, the count of triangle has some difference. This is because the Lanczos algorithm we implemented failed to find out all the greatest eigenvalue.
But they all have the same pattern, that is, subject to the power law. There are a great many of nodes that have small number of triangle and a few number of nodes that have very large number of triangle.
From the computation from MATLAB, we can see that there are about 10 nodes with extremely large number of triangles, that maybe the super star on youtube website.
And there are stand out phenomenon at around triangle number 100. This may because there are certain youtube groups of people that they all know each other in the same group.

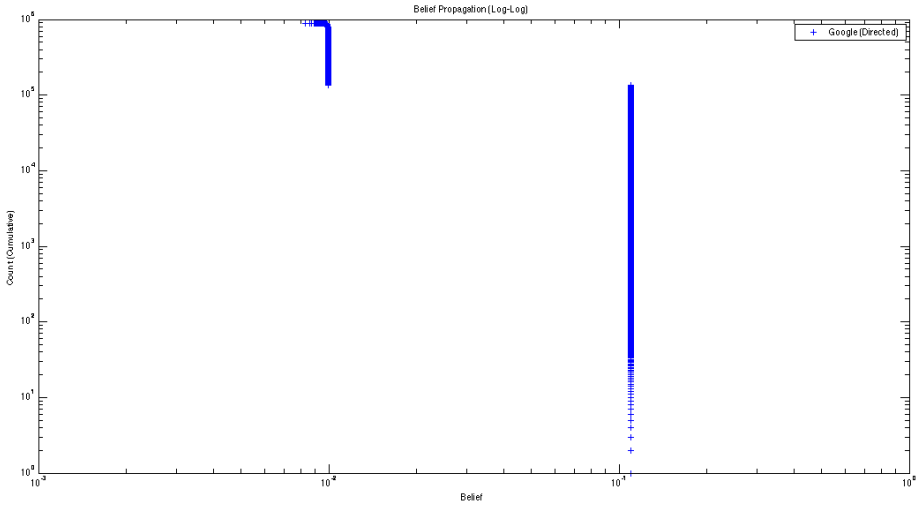
4.8.2 Variety Demonstration

We perform the test on 1M nodes social-Youtube dataset. From 18, we observes the triangle partic- ipation law (TPL).

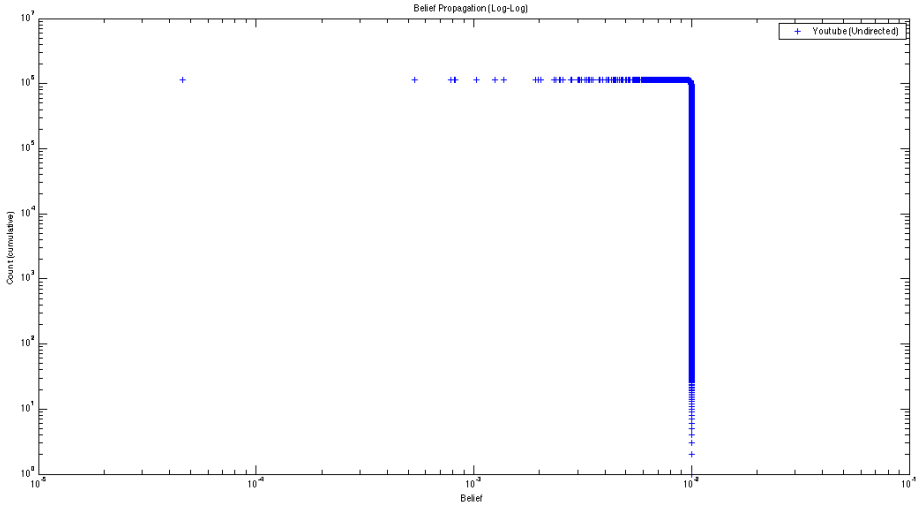
4.9 Task 8: Innovation Task

As we mentioned in the method section, we intend to extend algorithms to directed graph and weighted graph. As for directed graph, we have alread given the plot of web-Google graph in the variety sections of abbove related tasks.
Here we will cover the case of weighted degree distribution and weighted PageRank distribution. The graph we use is Libimseti dataset.
First we give the plot of in/out-degree and in/out weight as follows in Figure 19.
Then we plot in-weight and in-degree together as well as out-weight and out-degree together in Figure 20, which shows the SNAPSHOT POWER LAWS (SPL).

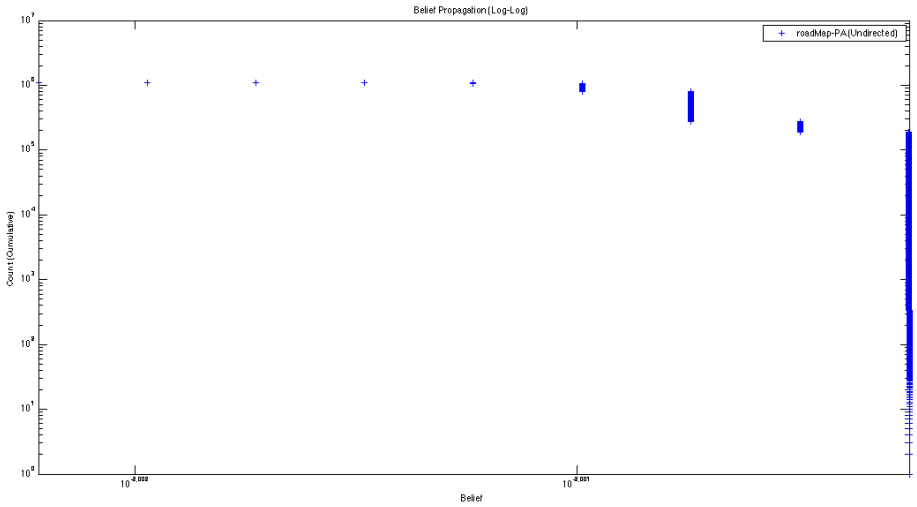
Finally, we give the plot of PageRank distribution and distributed PageRank distribution. From Figure 21, we can figure out that both of graphs follow the power law distribution and they are very similar to each other.



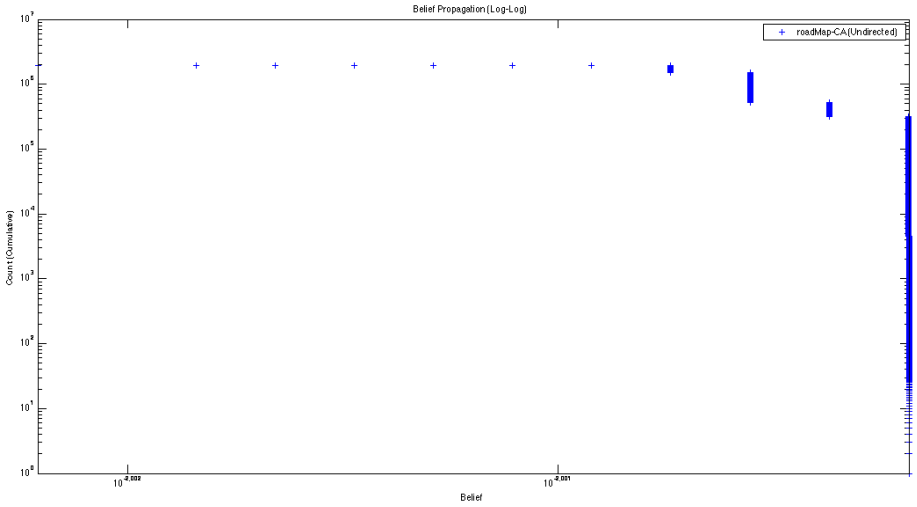
(a) Google



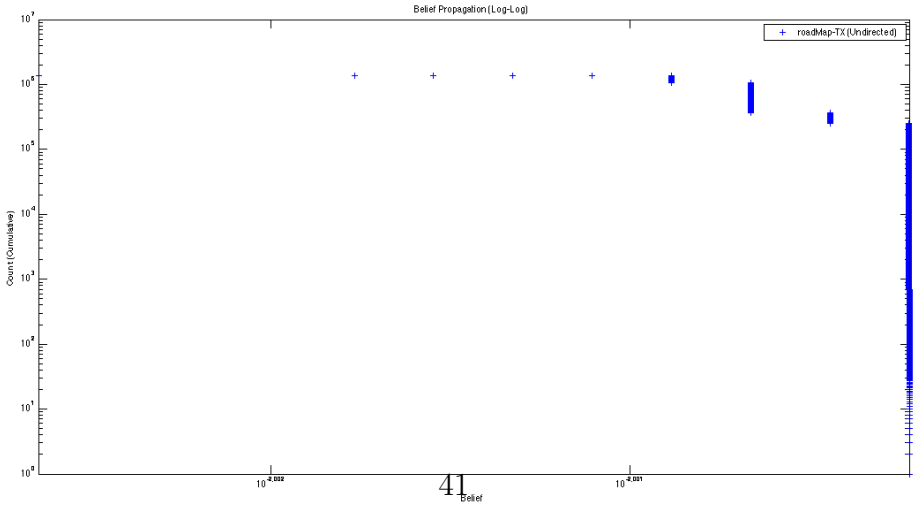
(b) Youtube



(a) roadMap-PA

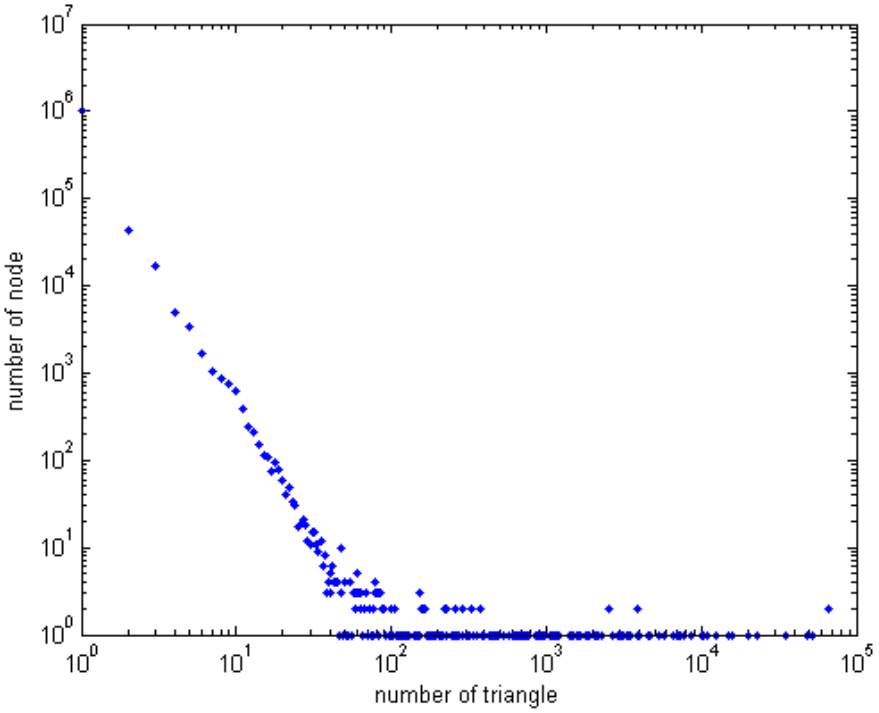


(b) roadMap-CA

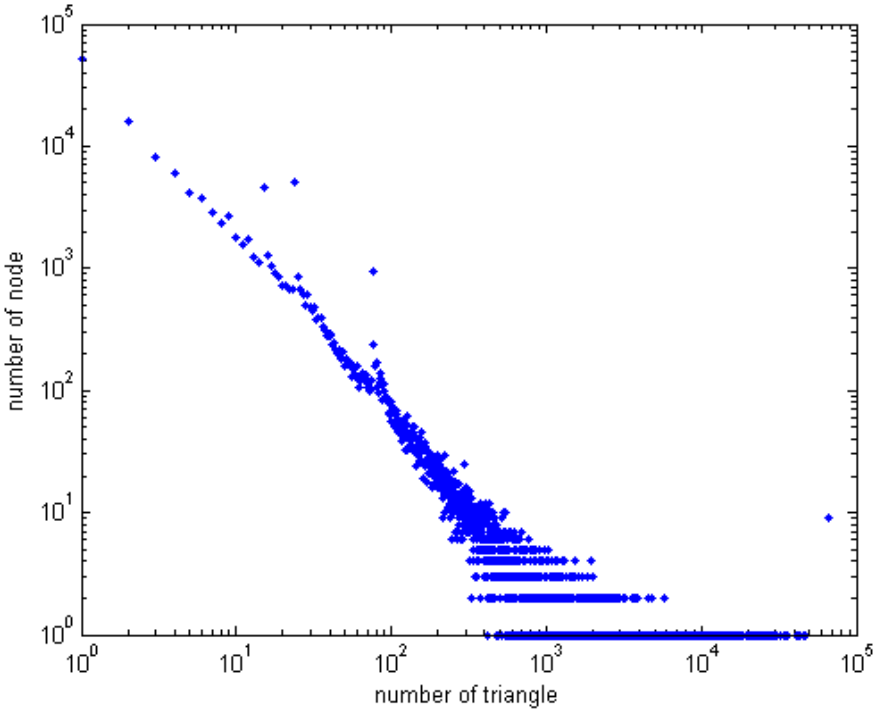


(c) roadMap-TX

Figure 17: Belief Distribution

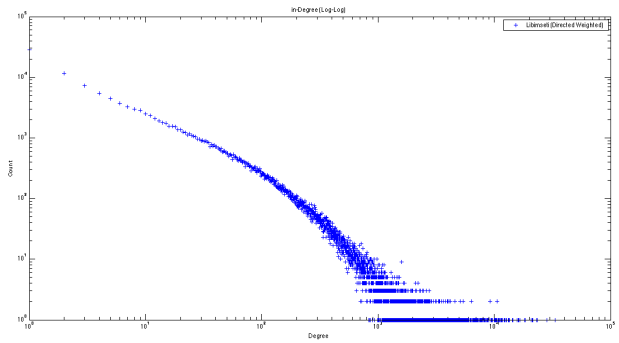


(a) Youtube: Triangle Distribution

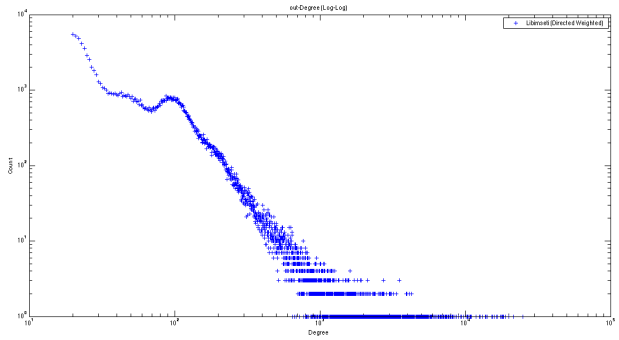


(b) Youtube: Triangle Distribution(Matlab)

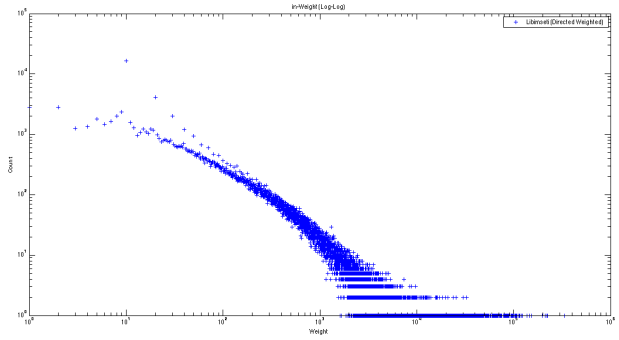
Figure 18: Weight/Triangle Distribution



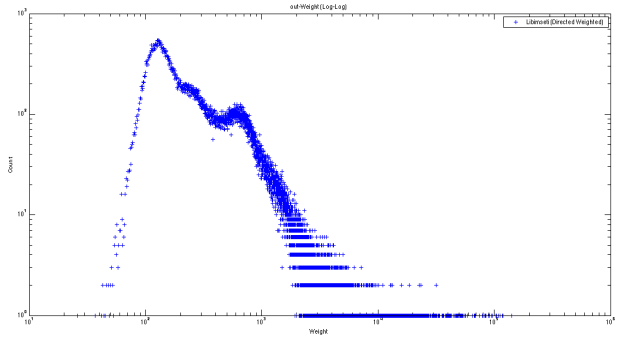
(a) Libimseti in-Degree



(b) Libimseti out-Degree

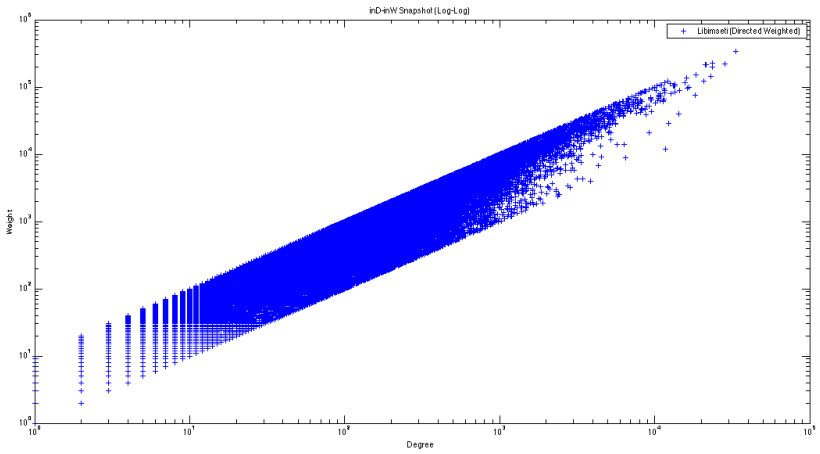


(c) Libimseti in-Weight

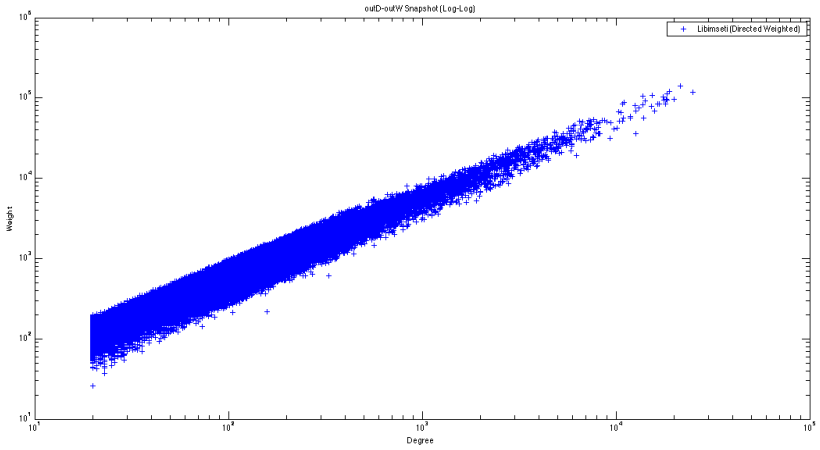


(d) Libimseti out-Weight

Figure 19: Weight and Degree Distribution

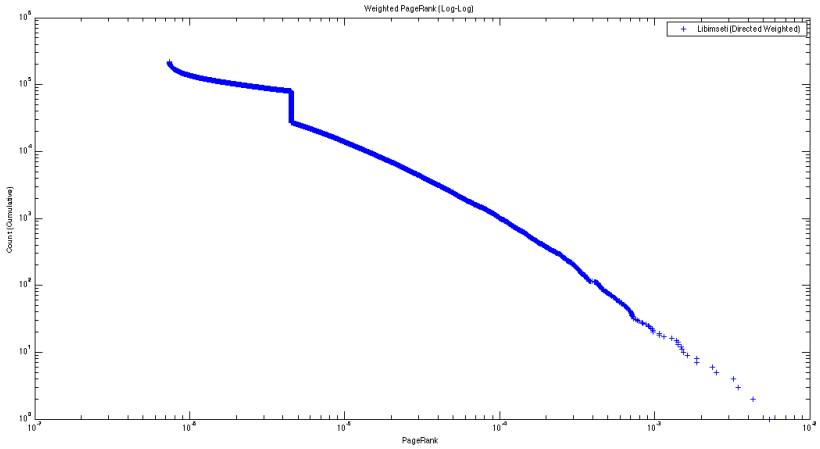


(a) inD-inW snapshot

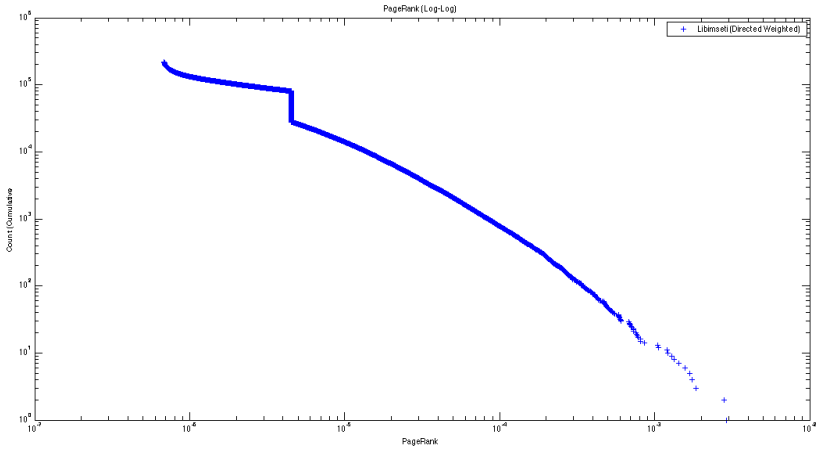


(b) outD-outW snapshot

Figure 20: Weight/Degree Snapshot



(a) Weighted PageRank Distribution



(b) PageRank Distribution

Figure 21: PageRank vs Weighted PageRank

5 Conclusions

In this report, we studied how to perform graphing mining in RDBMS. Implementating graph queries using SQL has the following advantages:

- it avoids the extra effort of extracting data and performing analysis elsewhere
- it makes full use of the support of RDBMS(e.g. JOIN operations, data management, query optimization, etc.)

After performing graph mining on various large-scale dataset, we come to the conclusions:

- real world large graph data are skewed distributed, and thus power lawer distribution are applied at many places
- large graph in information technology area such as graph social network and web graph behave very differently from real work network like road network in terms of small diameter

References

[1] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Comput. Netw. ISDN Syst.*, 30(1-7):107–117, April 1998.

[2] Philippe Flajolet and G. Nigel Martin. Probabilistic counting algorithms for data base applications. *J. Comput. Syst. Sci.*, 31(2):182–209, September 1985.

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

[4] U. Kang, B. Meeder, E. Papalexakis, and C. Faloutsos. Heigen: Spectral analysis for billion-scale graphs. *Knowledge and Data Engineering, IEEE Transactions on*, PP(99):1, 2012.

[5] U. Kang, Hanghang Tong, Jimeng Sun, Ching-Yung Lin, and Christos Faloutsos. Gbase: an efficient analysis platform for large graphs. *VLDB J.*, 21(5):637–650, 2012.

[6] U. Kang, Charalampos E. Tsourakakis, Ana Paula Appel, Christos Faloutsos, and Jure Leskovec. Hadi: Mining radii of large graphs. *ACM Trans. Knowl. Discov. Data*, 5:8:1–8:24, February 2011.

[7] U. Kang, Charalampos E. Tsourakakis, and Christos Faloutsos. Pegasus: mining peta-scale graphs. *Knowl. Inf. Syst.*, 27(2):303–325, 2011.

[8] Danai Koutra, Tai-You Ke, U Kang, Duen Horng Polo Chau, Hsing-Kuo Kenneth Pao, and Christos Faloutsos. Unifying guilt-by-association approaches: Theorems and fast algorithms. In *Proceedings of the ECML/PKDD 2011*, 2011.

[9] 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 *SIGMOD '09: Proceedings of the 35th SIGMOD international conference on Management of data*, pages 165–178, New York, NY, USA, 2009. ACM.

[10] C.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, 2008.

A Appendix

A.1 Labor Division

The team performed the following tasks:

- Implementation of degree distribution [Emma, Fangyu]
- Implementation of PageRank [Emma, Fangyu]
- Implementation of (weakly) connected components [Emma, Fangyu]
- Implementation of radius and diameter query [Emma]
- Implementation of eigenvalues/singular values [Fangyu]
- Implementation of Belief Propagation [Emma]
- Implementation of Count of Triangles [Fangyu]
- Experiments on the real data (Broad-spectrum graph mining) [Emma, Fangyu]
- Innovation tasks [Emma, Fangyu]

A tentative progress plan for Emma Zhang:

- by 10/10: Implementation of degree distribution [Done]
- by 10/10: Implementation of PageRank [Done]
- by 10/10: Implementation of (weakly) connected components [Done]
- by 10/24: Implementation of radius and diameter query [Done]
- by 10/24: Implementation of Belief Propagation [Done]
- by 11/01: Broad-spectrum graph mining(phase 1) [Done]
- by 11/07: Innovation task proposal [Done]
- by 11/07: Progress Report Writing [Done]
- by 11/21: Implementation of Innovation task [Done]
- by 11/21: Broad-spectrum graph mining(phase 2) [Done]
- by 11/26: Code Packaging and Final Report Writing [Done]

A tentative progress plan for Fangyu Gao:

- 10/12-10/22: implement HEigen algorithm [Done]
- 10/23-10/25: implement counting triangles algorithm [Done]
- 10/26-11/01: implement counting triangles algorithm [Done]
- 11/02-11/07: write progress report [Done]
- 11/08-11/19: implement innovation projects [Done]
- 11/20-11/26: Package code and write final report [Done]

A.2 Full disclosure wrt dissertations/projects

Emma Zhang: She is not doing any project or dissertation related to this project.

Fangyu Gao: He is not doing any project or dissertation related to this project.

A.3 Code Description

One thing we insist on during our processing is the making the code we write to a software that is robustness and easy to use.
Therefore, we write functions using PL/pgSQL that defaultly comes with PostgresSQL. We break the tasks in our project into small pieces and write them in separate functions. This guarantee the reusable and readability of our software.
We use table name as input and output parameters when passing tables, this makes our codes easy to understand and convenient to use. For example, the function matrix_multiply_matrix is in the following format:

matrix_by_matrix(dst_matrix_name, left_matrix_name, right_matrix_name) return void

If we want to calculate $A * B$ and put the result into C, that is $C = A * B$ we can use the statement:

matrix_by_matrix(C, A, B)

where A, B, C are the names of the three matrices. We always put the name of the return table in the first parameter of a function. This makes our software follow the usual coding style that output comes first(e.g. $C = A * B$). And by following this style, we make our functions easy to remember and convenient to use.
Table 4 lists the important functions we wrote have writen by now.

Function name	Description
void matrix_by_matrix(A, B, C)	$A = B * C$
void matrix_by_vector(a, A, b)	$a = A * b$
void matrix_transpose(A, B)	$A = B^T$
double vector_by_vector(a, b)	$a^T * b$
void vector_scale(a, b, lambda)	$a = lambda * b$

Table 4: Basic Functions

Contents