# Graph Mining Using SQL

*Ye Zhou*
School of Computer Science
CMU
yezhou@cs.cmu.edu

*Jin Hu*
School of Computer Science
CMU
jinh@cs.cmu.edu

December 2, 2014

**Abstract**

In this project, we investigated the possibility of using only SQL to do graph manipulations on *graphMiner*. *graphMiner* contains generic methods for RDBMS manipulations and some graph mining algorithms such as PageRank, Degree Distribution, Weak Connected Components, etc. Many useful properties of graphs can be found using these methods. *graphMiner* is based on Python, Postgres Database and SQL. Python provides basic control and logic as well as file operations, while all the important calculations are done through SQL on Postgres Database. We also implemented kcore algorithm within the framework, which gives useful information of the importance of nodes in the graph. We created unit tests on kcore and other algorithms in the system to test their functionality and correctness as well as to gain better understanding of these methods. For real world testing, we also ran kcore and other algorithms in the framework on 20 realworld datasets. We found some interesting properties of these datasets and explained them with the findings from the results of these graph mining algorithms.

## 1 Introduction

*Specify the problem; Give the motivation; List your main contributions*

The problem we are trying to solve is: Given a graph of source-destination pairs on disk, we try to use only SQL to implement basic matrix and vector operations, which make it possible to handle more complex graph manipulations. We will then be able to implement algorithms like PageRank and Degree distribution without the need of an additional language.

The motivation is that we can use only SQL and do not need to depend on other languages. With SQL we can enjoy the benefits of various features of databases and become more effcient in implementing and testing some of the algorithms without the worries like data structure, memory optimization, file IO, etc.

Doing graph mining with SQL is an important method for all scales, research or industry use, as database provides easier interface to scale and deploy. And graph manipulation and mining has many real world applications and is a very important topic. Thus a convenient way of implementing graph manipulation algorithms are highly desired.

The contributions of this project are the following:

- Our implemented *K-core Decomposition* is fast and uses only SQL to do the calculations. Python only works as some basic loop control and input output format handling.
- We tested various graph handling algorithm on unit tests and some real world datasets.

# 2 Survey

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

## 2.1 Papers read by Ye Zhou

The first paper was the Pegasus paper by U Kang

- *Main idea*: When graph data grows larger and larger, using traditional graph mining algorithms is difficult to deal with trend. In order to solve the graph mining problems with several Petabytes data, Pegasus is the first such library, impelemted on the top of Hadoop platform. In the paper, first they try to find out the common operation, which is the matrix-vector multipilication, underlying several primitive graph mining operations. They call it GIM-V. As GIM-V is so important, they successfully proposed several optimizations, and got more than 5 times faster performance. They also took real big data graph into Pegasus to get the mining result, which revealed important patterns. This showed the succuess of Pegasus with large data graph mininig as the graph data they used had never been studied before.
- *Use for our project*: It showed that with large data, we always have new problems using traditional graph mining methods. It is really hard to say that with SQL, we can do enough with graph mining now, as SQL is based on RMDB. But with more and more new mature products/framework like hive, pig, shark which support traditional SQL operations on big data and No-SQL database, we can do more with SQL.
- *Shortcomings*: PEGASUS focus on large graph querying/mining and most of the job was focused on how to compute fast, but it ignored the storage part which can also take effect to improve the performance like indexing. In addition, PEGASUS essentially perform node/vertex-centralized computation but cannot supports edge-centralized processing like induced subgraphs. Finally, hadoop is not so efficient for iterative calculation, as everytime it needs to write output data to hard disk. Spark has better performance as it output its intermediate data in memory and can even cahche the data in memory.

## 2.2 Papers read by Ye Zhou

The second paper was the Spectral Analysis for Billion-Scale Graphs paper by U Kang

- *Main idea*: The paper proposed HEIGEN algorithm which is designed to be accurate, efficient to run on highly scalable hadoop environment and solve the problems that will calculate out the spectral value. The paper first showed specific observation using HEIGEN with real world data on billion-scale graphs, focusing on structural property of networks: spotting near-cliques and finding triangles. Then the author explained that the alternatives for computing the eigenvalues of symmetric matrix including Power Method, Simultaneous iteration and Lanczos-NO are not suitable for big data on mapreduce. So the author described the algorithm for computing the top K eigenvectors and eigenvalues with four speficic fields improvement: Careful Algorithm Choice, selective parallelization, blocking and skewness exploiting. Finally it turned out that performance improved in both scalabitily and skewed matrix data, compared with HEIGEN-PLAIN.
- *Use for our project*: Largely based on mapreduce, HEIGEN is a totally new algorithm. What we can learn is that with massive data, we can change the former way of thinking for graph mining, so that we can largely improve the performance without SQL. And also, with the infrastructure of hadoop, and the way map/reduce reading data, we can do modification to use the advantages to get even better performance.
- *Shortcomings*: Highly based on map/reduce architecture also brings lots of problems that hadoop has. Such as the job schedualing and data shuffling. As the matrix operation needs to read large amount of data, and for iterative calculation, spark is a better choice as everything is in memory.

## 2.3 Papers read by Ye Zhou

The third paper was the Unifying Guil-by-Association Approaches paper by Koutra

- *Main idea*: The paper mainly proposed FaBP, which is a Fast Belief Propagation algorithm on Hadoop. It first compare and contrast several very successful, guilt-by-association methods: Random Walk with Restarts, Semi-Supervised Learning and Belief Propagation. The author showed that these three methods are closedly related but not identical. Then he proposed the algorithm FaBP, showed the experiments result. It turned out that the accuracy keeps the same or even better with the traditional BP, but the performance is twice better. It also has convergence guarantee. It is even sensitive to the "about-half" homophily factor, as long as the latter is within the convergence bounds. It also scales linearly on the number of edges.
- *Use for our project*: Learn the way using hadoop to impelment the algorithm for machine learning.
- *Shortcomings*: Again it is based on Hadoop, the performance for iterative calculation is not so good compared with spark. And BP algorithm is not so efficient when dealing with graph which has circle. And the convergence is limited due to specific requirement.

## 2.4 Papers read by Jin Hu

The first paper was the GBASE paper by U Kang

- *Main idea*: The paper introduces a general graph management system GBase for large scale graph storage and computation.
- *The main contribution of the paper*:: GBase uses "compressed block encoding" method to make graph storage more efficiently. For graph indexing, the paper succeeds in handling multiple type of queries on a large graph instead of a specific type and is suitable for distributed environment. By supporting homogeneous block level indexing and being flexible in both edge and node centralized computing, GBase has better properties than similar distributed systems. The framework the paper proposes also supported both graph-level and node-level queries, making it applicable to various applications. GBase partitions data in two dimensions to better use the block and community-like properties of real-world graphs, which gives it advantage over either row-oriented or column-oriented storages.
- *Limitations*: The paper's indexing method handles large graphs successfully, but its property compared to frequent subgraph or significant graph pattern methods are not shown in the experiment. Optional indexing methods may be added to the system.

## 2.5 Papers read by Jin Hu

The second paper was by Danai Koutra

- *Main idea*: The paper does the comparison among some of the most popular guilt-by-association method.
- *The main contribution of the paper*:: The paper manages to prove that all methods result in a similar matrix inversion problem. In addition, the paper proposes a fast and accurate BP algorithm. In theory, the paper finds that RWR(Personalized Random Walk with Restats), SSL(Semi-Supervised Learning) and BP(Belief Propagation) are closely related, but not the same. RWR and SSL are not heterophily, but BP is heterophily. All three methods are scalable. RWR and SSL have convergence while BP is unknown. The proposed FABP method has nice property with all these perspectives. FABP is an approximation of standard BP, but FABP is significantly faster based on the experiment and guarantees convergence, which makes it better than BP. The experiments also verify the paper's ideas. The author tested the theory and the properties of the proposed FABP method in terms of accuracy, convergence, sensitivity to parameters and scalability.

## 2.6 Papers read by Jin Hu

The third paper was by Ignacio Alvarez-Hamelin

- *Main idea*: The paper introduces K-core decomposition and its application in the visulization of large scale networks.

- *The main contribution of the paper*:: K-core decomposition can find subgraphs which all of the nodes in the subgraph have degrees higher than k after removing nodes with lower coreness. This method can find the subgraphs which are more closely connected and achieves "clustering" in large graphs. As K-core decomposition can produce two-dimensional layout of large scale networks with their important topological and hierarchical properties, the paper takes advantage of the K-core algorithm to allow visulization of network and offer features like fingerprint identification and general analysis assistance. The visualization algorithm has linear running time proportional to the size of the network, making it well scalable for large networks. In addition, the algorithm offers 2D representation of networks which makes information visualization more accessible than other representations and the parameters of the algorithm are universally defined, which makes it suitable for all types of networks.
- *Limitations*: The proposed visualization algorithm still utilizes certain parameters to identify the properties of the network, which involves considerable human interactions and prior expemental knowledge. Self adjusting parameters might be a huge improvement and can be an interesting topic to follow.

...

# 3 Proposed Method

We implemented K-core in *graphMiner* framework with SQL following the Batagelj and Zaversnik k-core variation [5]. We recursively prune the nodes and edges in the graph with degree less than k to finally arrive at points of degree greater than or equal to k. That is, in each iteration, we add nodes whose total indegree do not satisfy the iteration i requirement, i.e. the nodes that have insufficient nodes connected to them in each iteration are deleted. After iterations, all the remaining nodes satisfy K-core requirements and we do a Connected Component Analysis to find the clusters.

## 3.1 K-core Decomposition: Definition

A graph G = (V, E), $|V|$ = n vertices and $|E|$ = e edges. Then a K-core is a subgraph of $H = (C, E|C)$ induced by the set $C \subseteq Vif \forall v \in$ C: $degree_H(v) >= k$, and H is the maximum subgraph with this property.

K-cores in the graph can then be calculated by recursively removing all the vertices of degree less than k, until all vertices in the remaining graph have at least degree k.

A vertex i has coreness c if it belongs to the c-core but not to (c + 1)-core. We denote by $c_i$ the coreness of vertex i.

A shell $C_c$ is composed by all the vertices whose coreness is c. The maximum value c such that $C_c$ is not empty is denoted $c_max$. The k-core is thus the union of all shells $C_c$ with c  k.

Each connected set of vertices having the same coreness c is a cluster $Q^c$. Each shell $C_c$ is thus composed by clusters $Q^c_m$ such that $C_c = U_(1mq_m ax^c)Q^c_m$, where $q^c_m ax$ is the number of clusters in $C_c$.

### 3.2 K-core Algorithm Description

**Algorithm 1** K-core algorithm Description with SQL

**Require:** *UndirectGraph* and *NodeDegrees* are input data and prerequisite calculation, $k$ is the number of iterations and the k in K-core algorithm.

1: **function** KCORE(*UndirectGraph*, *NodeDegrees*, $n$)
2: **for** $i \leftarrow 1$ to $k$ **do**
create TempNodeDegrees, TempUndirectGraph table that do not contain nodes in the RemovedNodeTable
INSERT INTO RemovedNodeTable SELECT nodeId FROM TempNodeDegrees WHERE inDegree ¡ i
Do Connected Components Analysis and return distinct componentId in Connected-ComponentGraph
3: Save Result to CSV File

### 3.3 K-core Implementation User Mannual

The minimum command to run the k-core algorithm in the *graphMiner* framework is as follows.

python gm_main.py ¡input graph file¿ ¡output graph file¿ ¡weighted flag¿ ¡directed flag¿

One concrete example:

python gm_main.py –file /your-path/input-graphfile –dest_dir /your-path/output –unweighted –undirected

More details on the command arguments, please refer to the readme file in the project directory.

Then the graphminer will print out the results for k-core in terminal and output K-core nodes and their corresponding component id in csv file in the output directory you designate. More details on the command arguments, please refer to the readme file in the project directory.

## 4 Experiments

We implemented kcores method and tested other algorithms that are already implemented in the system with unit tests and other datasets from SNAP.

## 4.1 Phase 1 Experiments

Figures below give the degree distribution and pagerank result of two dataset from SNAP. We can find that the degree distribution and pagerank results are consistent with the power law as nodes or pages with higher degree or rank have a small number while nodes or pages with lower degree or rank have a large number. You can also find the detailed results in the output folder, which contains csv files for the results of belief propagation, connected components, node degrees, degree distribution, eigen values, k-core connected components, pagerank results, radius, etc.

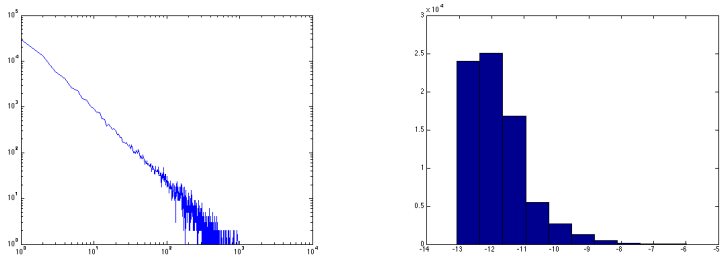
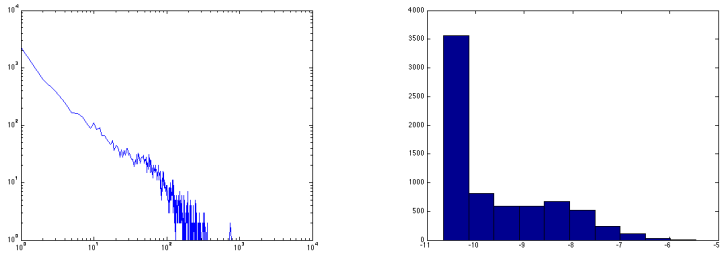


Figure 1: Degree Distribution(a) and PageRank(b) for Dataset SOC-Epinions1



(a) (b)

Figure 2: Degree Distribution(a) and PageRank(b) for Dataset wiki-Vote

The figures below also include the degree distribution, connected components, k=5 cores algorithm results on the 5 unit tests. As the unit tests are small, there is no nodes in the tests that satisfy k=5 cores, so the output result for these 5 unit tests are empty. However, you can find the node id, component id pairs in the stdout output from console or in the kcorecomponent.csv file, which shows that the k-core algorithm works as it claims to find correct coreness subgraphs.

| Degree | Count |
|---|---|
| 2 | 11 |
| | |

| Node_id | Component_id |
|---|---|
| 8 | 0 |
| 4 | 0 |
| 1 | 0 |
| 5 | 0 |
| 3 | 0 |
| 0 | 0 |
| 10 | 0 |
| 9 | 0 |
| 6 | 0 |
| 2 | 0 |
| 7 | 0 |

Figure 3: Degree Distribution(a), connected components(b) for Dataset1

| Degree | Count |
|---|---|
| 4 | 5 |

| Node_id | Component_id |
|---|---|
| 4 | 0 |
| 1 | 0 |
| 3 | 0 |
| 0 | 0 |
| 2 | 0 |

Figure 4: Degree Distribution(a), connected components(b) for Dataset2

| Degree | Count |
|---|---|
| 4 | 7 |
| 3 | 3 |
| 9 | 1 |

| Node_id | Component_id |
|---|---|
| 8 | 0 |
| 4 | 0 |
| 1 | 0 |
| 5 | 0 |
| 3 | 0 |
| 0 | 0 |
| 10 | 0 |
| 9 | 0 |
| 6 | 0 |
| 2 | 0 |
| 7 | 0 |

Figure 5: Degree Distribution(a), connected components(b) for Dataset3

| Degree | Count |
|---|---|
| 1 | 2 |
| 2 | 3 |

| Node_id | Component_id |
|---|---|
| 4 | 2 |
| 1 | 0 |
| 3 | 2 |
| 0 | 0 |
| 2 | 2 |

Figure 6: Degree Distribution(a), connected components(b) for Dataset4

| Degree | Count |
|---|---|
| 3 | 16 |
| 2 | 5 |

| Node_id | Component_id |
|---|---|
| 8 | 0 |
| 16 | 0 |
| 15 | 0 |
| 4 | 0 |
| 20 | 0 |
| 1 | 0 |
| 13 | 0 |
| 5 | 0 |
| 11 | 0 |
| 3 | 0 |
| 14 | 0 |
| 17 | 0 |
| 0 | 0 |
| 19 | 0 |
| 12 | 0 |
| 10 | 0 |
| 18 | 0 |
| 9 | 0 |
| 6 | 0 |
| 2 | 0 |
| 7 | 0 |

Figure 7: Degree Distribution(a) connected components(b) for Dataset5

## 4.2 Phase 2 Indexing Experiments

We build index for algorithms Degree Distribution, K-core, Pagerank, Connected Components, All Radius, Eigen Value Computation on ten datasets:as-skitter.ungraph-75000, ca-AstroPh, cit-HepPh, cit-HepTh, com-amazon.ungraph-75000, com-dblp.ungraph-75000, email-Enron.ungraph, email-EuAll,p2p-Gnutella31, soc-Slashdot0811-75000.

We first conducted some simple tests to make sure that creating index can lead to performance improvement. For example, Degree Distribution:(no index on node degree), run time is 147.476911545 Degree Distribution:(index on node degree (in degree, out degree)), run time is 346.571922302 From the experiment, create index does consume system resources and it will make performance worse in general if we created index and used it only once. However, Degree Distribution:(no index on node degree)(run 10 times), run time is 2091.14193916, Degree Distribution:(index on node degree (in degree, out degree))(run 10 times, 1 time create index), run time is 1351.35889053. From the experiment, create index will improve performance in general if we created index and used it later a lot. Also, for a certain algorithm, create index for some tables like GMNODES are expensive, but if re run all the algorithms,or use the algorithm multiple times, then the cost of creating index on tables like GMNODES will be worth the effort.

This is the general intuitive for us to choose on which table and which column to create index and avoid some unnecessay tests.

We compared building index for different tables on different columns for each algorithm and get the following figures below.

| All Radius | | | | Combination Choice | |
|---|---|---|---|---|---|
| as-skitter.ungraph-75000.txt | Create Time | Whole Time | | | Whole Time |
| GM_TABLE_UNDIRECT (src_id) | 117 | 94563 | ✖ | NoIndex | 91840 |
| prev_hop_table (node_id) | 115 | 95170 | ✖ | WithBestIndex | ✖ |
| max_hop_ngh (id) | 108 | 94377 | ✖ | | |
| | | | | | |
| ca-AstroPh.txt | Create Time | Whole Time | | | Whole Time |
| GM_TABLE_UNDIRECT (src_id) | 37 | 22040 | ✖ | NoIndex | 20684 |
| prev_hop_table (node_id) | 42 | 22639 | ✖ | WithBestIndex | ✖ |
| max_hop_ngh (id) | 91 | 32491 | ✖ | | |
| | | | | | |
| cit-HepPh.txt | Create Time | Whole Time | | | Whole Time |
| GM_TABLE_UNDIRECT (src_id) | 58 | 30688 | ✖ | NoIndex | 31902 |
| prev_hop_table (node_id) | 60 | 31537 | ✖ | WithBestIndex | ✖ |
| max_hop_ngh (id) | 68 | 26540 | ✖ | | |
| | | | | | |
| cit-HepTh.txt | Create Time | Whole Time | | | Whole Time |
| GM_TABLE_UNDIRECT (src_id) | 54 | 25942 | ✖ | NoIndex | 25025 |
| prev_hop_table (node_id) | 53 | 25346 | ✖ | WithBestIndex | ✖ |
| max_hop_ngh (id) | 63 | 23666 | ✖ | | |
| | | | | | |
| com-amazon.ungraph-75000.txt | Create Time | Whole Time | | | Whole Time |
| GM_TABLE_UNDIRECT (src_id) | 110 | 145466 | ✖ | NoIndex | 141871 |
| prev_hop_table (node_id) | 119 | 145499 | ✖ | WithBestIndex | ✖ |
| max_hop_ngh (id) | 150 | 148162 | ✖ | | |
| | | | | | |
| com-dblp.ungraph-75000.txt | Create Time | Whole Time | | | Whole Time |
| GM_TABLE_UNDIRECT (src_id) | 112 | 87245 | ✖ | NoIndex | 82507 |
| prev_hop_table (node_id) | 121 | 89466 | ✖ | WithBestIndex | ✖ |
| max_hop_ngh (id) | 128 | 88248 | ✖ | | |
| | | | | | |
| email-Enron.ungraph.txt | Create Time | Whole Time | | | Whole Time |
| GM_TABLE_UNDIRECT (src_id) | 60 | 30373 | ✖ | NoIndex | 28366 |
| prev_hop_table (node_id) | 67 | 30541 | ✖ | WithBestIndex | ✖ |
| max_hop_ngh (id) | 106 | 29462 | ✖ | | |
| | | | | | |
| email-EuAll.txt | Create Time | Whole Time | | | Whole Time |
| GM_TABLE_UNDIRECT (src_id) | 412 | 155245 | ✖ | NoIndex | 149123 |
| prev_hop_table (node_id) | 347 | 158208 | ✖ | WithBestIndex | ✖ |
| max_hop_ngh (id) | 486 | 152816 | ✖ | | |
| | | | | | |
| p2p-Gnutella31.txt | Create Time | Whole Time | | | Whole Time |
| GM_TABLE_UNDIRECT (src_id) | 94 | 48675 | ✖ | NoIndex | 46756 |
| prev_hop_table (node_id) | 114 | 49863 | ✖ | WithBestIndex | ✖ |
| max_hop_ngh (id) | 117 | 47568 | ✖ | | |
| | | | | | |
| soc-Slashdot0811-75000.txt | Create Time | Whole Time | | | Whole Time |
| GM_TABLE_UNDIRECT (src_id) | 113 | 54130 | ✖ | NoIndex | 53396 |
| prev_hop_table (node_id) | 196 | 57284 | ✖ | WithBestIndex | ✖ |
| max_hop_ngh (id) | 129 | 53106 | ✖ | | |



Figure 8: Creating Index Experiments on All Radius Algorithm

| Connected Component | | | | Combination Choice | |
|---|---|---|---|---|---|
| as-skitter.ungraph-75000.txt | Create Time | Whole Time | | | Whole Time |
| GM_TABLE_UNDIRECTED (node_id) | 846 | 29178 | ✔ | NoIndex | 31738 |
| GM_CC_TEMP (node_id) | 115 | 29616 | ✔ | WithBestIndex | 29560 |
| GM_TABLE_UNDIRECTED (node_id) | 113 | 29122 | ✔ | | |
| | | | | | |
| ca-AstroPh.txt | Create Time | Whole Time | | | Whole Time |
| GM_TABLE_UNDIRECTED (node_id) | 756 | 14033 | ✔ | NoIndex | 14231 |
| GM_CC_TEMP (node_id) | 35 | 13649 | ✔ | WithBestIndex | 13885 |
| GM_TABLE_UNDIRECTED (node_id) | 34 | 13513 | ✔ | | |
| | | | | | |
| cit-HepPh.txt | Create Time | Whole Time | | | Whole Time |
| GM_TABLE_UNDIRECTED (node_id) | 796 | 14657 | ✔ | NoIndex | 16102 |
| GM_CC_TEMP (node_id) | 58 | 13135 | ✔ | WithBestIndex | 16180 |
| GM_TABLE_UNDIRECTED (node_id) | 63 | 13723 | ✔ | | |
| | | | | | |
| cit-HepTh.txt | Create Time | Whole Time | | | Whole Time |
| GM_TABLE_UNDIRECTED (node_id) | 756 | 13606 | ✔ | NoIndex | 15178 |
| GM_CC_TEMP (node_id) | 47 | 13324 | ✔ | WithBestIndex | 14283 |
| GM_TABLE_UNDIRECTED (node_id) | 46 | 13168 | ✔ | | |
| | | | | | |
| com-amazon.ungraph-75000.txt | Create Time | Whole Time | | | Whole Time |
| GM_TABLE_UNDIRECTED (node_id) | 369 | 83850 | ✔ | NoIndex | 91874 |
| GM_CC_TEMP (node_id) | 114 | 80122 | ✔ | WithBestIndex | 79700 |
| GM_TABLE_UNDIRECTED (node_id) | 117 | 78914 | ✔ | | |
| | | | | | |
| com-dblp.ungraph-75000.txt | Create Time | Whole Time | | | Whole Time |
| GM_TABLE_UNDIRECTED (node_id) | 754 | 36159 | ✔ | NoIndex | 34391 |
| GM_CC_TEMP (node_id) | 115 | 29657 | ✔ | WithBestIndex | 31750 |
| GM_TABLE_UNDIRECTED (node_id) | 128 | 29727 | ✔ | | |
| | | | | | |
| email-Enron.ungraph.txt | Create Time | Whole Time | | | Whole Time |
| GM_TABLE_UNDIRECTED (node_id) | 1315 | 15936 | ✔ | NoIndex | 14354 |
| GM_CC_TEMP (node_id) | 60 | 15088 | ✔ | WithBestIndex | 15775 |
| GM_TABLE_UNDIRECTED (node_id) | 60 | 14259 | ✔ | | |
| | | | | | |
| email-EuAll.txt | Create Time | Whole Time | | | Whole Time |
| GM_TABLE_UNDIRECTED (node_id) | 1641 | 52163 | ✔ | NoIndex | 50984 |
| GM_CC_TEMP (node_id) | 213 | 47881 | ✔ | WithBestIndex | 49792 |
| GM_TABLE_UNDIRECTED (node_id) | 213 | 46924 | ✔ | | |
| | | | | | |
| p2p-Gnutella31.txt | Create Time | Whole Time | | | Whole Time |
| GM_TABLE_UNDIRECTED (node_id) | 496 | 11825 | ✔ | NoIndex | 12905 |
| GM_CC_TEMP (node_id) | 95 | 10701 | ✔ | WithBestIndex | 10795 |
| GM_TABLE_UNDIRECTED (node_id) | 98 | 11060 | ✔ | | |
| | | | | | |
| soc-Slashdot0811-75000.txt | Create Time | Whole Time | | | Whole Time |
| GM_TABLE_UNDIRECTED (node_id) | 1520 | 23275 | ✔ | NoIndex | 22870 |
| GM_CC_TEMP (node_id) | 126 | 20600 | ✔ | WithBestIndex | 22121 |
| GM_TABLE_UNDIRECTED (node_id) | 110 | 20316 | ✔ | | |

Figure 9: Creating Index Experiments on Connected Components Algorithm

| Degree Distribution | | | | Combination Choice | |
|---|---|---|---|---|---|
| as-skitter.ungraph-75000.txt | Create Time | Whole Time | | | Whole Time |
| GM_NODE_DEGREES (in_degree) | 118.96 | 210.53 | ✔ | NoIndex | 137.54 |
| GM_NODE_DEGREES (out_degree) | 170.25 | 298.91 | ✔ | With✔Index | 413.98 |
| | | | | | |
| ca-AstroPh.txt | Create Time | Whole Time | | | Whole Time |
| GM_NODE_DEGREES (in_degree) | 39.04 | 133.09 | ✔ | NoIndex | 87.94 |
| GM_NODE_DEGREES (out_degree) | 33.01 | 131.39 | ✔ | WithBestIndex | 169.04 |
| | | | | | |
| cit-HepPh.txt | Create Time | Whole Time | | | Whole Time |
| GM_NODE_DEGREES (in_degree) | 85.06 | 207.19 | ✔ | NoIndex | 540.9 |
| GM_NODE_DEGREES (out_degree) | 111.52 | 260.08 | ✔ | WithBestIndex | 265.35 |
| | | | | | |
| cit-HepTh.txt | Create Time | Whole Time | | | Whole Time |
| GM_NODE_DEGREES (in_degree) | 58.83 | 173.5 | ✔ | NoIndex | 739.02 |
| GM_NODE_DEGREES (out_degree) | 58.5 | 435.44 | ✔ | WithBestIndex | 194.34 |
| | | | | | |
| com-amazon.ungraph-75000.txt | Create Time | Whole Time | | | Whole Time |
| GM_NODE_DEGREES (in_degree) | 142.18 | 263.03 | ✔ | NoIndex | 125.27 |
| GM_NODE_DEGREES (out_degree) | 164.74 | 750.66 | ✔ | WithBestIndex | 352.07 |
| | | | | | |
| com-dblp.ungraph-75000.txt | Create Time | Whole Time | | | Whole Time |
| GM_NODE_DEGREES (in_degree) | 156.32 | 270.31 | ✔ | NoIndex | 110.62 |
| GM_NODE_DEGREES (out_degree) | 124.51 | 235.19 | ✔ | WithBestIndex | 2668.33 |
| | | | | | |
| email-Enron.ungraph.txt | Create Time | Whole Time | | | Whole Time |
| GM_NODE_DEGREES (in_degree) | 70.88 | 193.11 | ✔ | NoIndex | 101.54 |
| GM_NODE_DEGREES (out_degree) | 105.33 | 259.64 | ✔ | WithBestIndex | 250.97 |
| | | | | | |
| email-EuAll.txt | Create Time | Whole Time | | | Whole Time |
| GM_NODE_DEGREES (in_degree) | 426.1 | 619.24 | ✔ | NoIndex | 225.74 |
| GM_NODE_DEGREES (out_degree) | 1598.54 | 1791.75 | ✔ | WithBestIndex | 965.82 |
| | | | | | |
| p2p-Gnutella31.txt | Create Time | Whole Time | | | Whole Time |
| GM_NODE_DEGREES (in_degree) | 173.58 | 336.88 | ✔ | NoIndex | 138.97 |
| GM_NODE_DEGREES (out_degree) | 827.34 | 1417.15 | ✔ | WithBestIndex | 330.83 |
| | | | | | |
| soc-Slashdot0811-75000.txt | Create Time | Whole Time | | | Whole Time |
| GM_NODE_DEGREES (in_degree) | 171.23 | 1507.81 | ✔ | NoIndex | 137.84 |
| GM_NODE_DEGREES (out_degree) | 152.01 | 278.07 | ✔ | WithBestIndex | 317.91 |

Figure 10: Creating Index Experiments on Degree Distribution Algorithm

| Eigen | | | | Combination Choice | |
|---|---|---|---|---|---|
| as-skitter.ungraph-75000.txt | Create Time | Whole Time | | | Whole Time |
| % (EVal) + (row_id) | 5 | 66153 | ✖ | NoIndex | 69930 |
| % (EVal) + (col_id) | 5 | 70915 | ✖ | WithBestIndex | ✖ |
| % (basis_vect_0) + (id) | 139 | | ✖ | | |
| % (basis_vect_1) + (id) | 191 | 70338 | ✖ | | |
| | | | | | |
| ca-AstroPh.txt | Create Time | Whole Time | | | Whole Time |
| % (EVal) + (row_id) | 5 | 20034 | ✖ | NoIndex | 19574 |
| % (EVal) + (col_id) | 5 | 19624 | ✖ | WithBestIndex | ✖ |
| % (basis_vect_0) + (id) | 39 | | ✖ | | |
| % (basis_vect_1) + (id) | 52 | 20021 | ✖ | | |
| | | | | | |
| cit-HepPh.txt | Create Time | Whole Time | | | Whole Time |
| % (EVal) + (row_id) | 5 | 40535 | ✖ | NoIndex | 32798 |
| % (EVal) + (col_id) | 5 | 42522 | ✖ | WithBestIndex | ✖ |
| % (basis_vect_0) + (id) | 60 | | ✖ | | |
| % (basis_vect_1) + (id) | 88 | 37508 | ✖ | | |
| | | | | | |
| cit-HepTh.txt | Create Time | Whole Time | | | Whole Time |
| % (EVal) + (row_id) | 5 | 22843 | ✖ | NoIndex | 23979 |
| % (EVal) + (col_id) | 5 | 22116 | ✖ | WithBestIndex | ✖ |
| % (basis_vect_0) + (id) | 54 | | ✖ | | |
| % (basis_vect_1) + (id) | 70 | 22393 | ✖ | | |
| | | | | | |
| com-amazon.ungraph-75000.txt | Create Time | Whole Time | | | Whole Time |
| % (EVal) + (row_id) | 5 | 67768 | ✖ | NoIndex | 77713 |
| % (EVal) + (col_id) | 5 | 70561 | ✖ | WithBestIndex | ✖ |
| % (basis_vect_0) + (id) | 123 | | ✖ | | |
| % (basis_vect_1) + (id) | 170 | 58789 | ✖ | | |
| | | | | | |
| com-dblp.ungraph-75000.txt | Create Time | Whole Time | | | Whole Time |
| % (EVal) + (row_id) | 5 | 31416 | ✖ | NoIndex | 29544 |
| % (EVal) + (col_id) | 5 | 30085 | ✖ | WithBestIndex | ✖ |
| % (basis_vect_0) + (id) | 79 | | ✖ | | |
| % (basis_vect_1) + (id) | 239 | 30129 | ✖ | | |
| | | | | | |
| email-Enron.ungraph.txt | Create Time | Whole Time | | | Whole Time |
| % (EVal) + (row_id) | 5 | 18978 | ✖ | NoIndex | 18466 |
| % (EVal) + (col_id) | 5 | 19505 | ✖ | WithBestIndex | ✖ |
| % (basis_vect_0) + (id) | 61 | | ✖ | | |
| % (basis_vect_1) + (id) | 87 | 18322 | ✖ | | |
| | | | | | |
| email-EuAll.txt | Create Time | Whole Time | | | Whole Time |
| % (EVal) + (row_id) | 5 | 56270 | ✖ | NoIndex | 57341 |
| % (EVal) + (col_id) | 5 | 54228 | ✖ | WithBestIndex | ✖ |
| % (basis_vect_0) + (id) | 454 | | ✖ | | |
| % (basis_vect_1) + (id) | 321 | 55777 | ✖ | | |
| | | | | | |
| p2p-Gnutella31.txt | Create Time | Whole Time | | | Whole Time |
| % (EVal) + (row_id) | 5 | 368635 | ✖ | NoIndex | 90771 |
| % (EVal) + (col_id) | 5 | 244377 | ✖ | WithBestIndex | ✖ |
| % (basis_vect_0) + (id) | 141 | | ✖ | | |
| % (basis_vect_1) + (id) | 214 | 267490 | ✖ | | |

| K-core | | | | Combination Choice | |
|---|---|---|---|---|---|
| as-skitter.ungraph-75000.txt | Create Time | Whole Time | | | Whole Time |
| GM_TABLE_UNDIRECTED (dst_id) | 382 | 14694 | | NoIndex | 17471 |
| TEMP_GM_TABLE_UNDIRECT (src_id) | 63 | 16723 | | WithBestIndex | 16355 |
| TEMP_GM_TABLE_UNDIRECT (dst_id) | 62 | 14045 | | | |
| | | | | | |
| ca-AstroPh.txt | Create Time | Whole Time | | | Whole Time |
| GM_TABLE_UNDIRECTED (dst_id) | 703 | 28664 | | NoIndex | 28359 |
| TEMP_GM_TABLE_UNDIRECT (src_id) | 19 | 27934 | | WithBestIndex | 32589 |
| TEMP_GM_TABLE_UNDIRECT (dst_id) | 823 | 27822 | | | |
| | | | | | |
| cit-HepPh.txt | Create Time | Whole Time | | | Whole Time |
| GM_TABLE_UNDIRECTED (dst_id) | 805 | 28982 | | NoIndex | 31040 |
| TEMP_GM_TABLE_UNDIRECT (src_id) | 28 | 30986 | | WithBestIndex | 33887 |
| TEMP_GM_TABLE_UNDIRECT (dst_id) | 712 | 27733 | | | |
| | | | | | |
| cit-HepTh.txt | Create Time | Whole Time | | | Whole Time |
| GM_TABLE_UNDIRECTED (dst_id) | 637 | 25921 | | NoIndex | 21903 |
| TEMP_GM_TABLE_UNDIRECT (src_id) | 24 | 22678 | | WithBestIndex | 26026 |
| TEMP_GM_TABLE_UNDIRECT (dst_id) | 28 | 24197 | | | |
| | | | | | |
| com-amazon.ungraph-75000.txt | Create Time | Whole Time | | | Whole Time |
| GM_TABLE_UNDIRECTED (dst_id) | 355 | 34682 | | NoIndex | 34847 |
| TEMP_GM_TABLE_UNDIRECT (src_id) | 89 | 37735 | | WithBestIndex | 38220 |
| TEMP_GM_TABLE_UNDIRECT (dst_id) | 101 | 33985 | | | |
| | | | | | |
| com-dblp.ungraph-75000.txt | Create Time | Whole Time | | | Whole Time |
| GM_TABLE_UNDIRECTED (dst_id) | 401 | 22832 | | NoIndex | 21399 |
| TEMP_GM_TABLE_UNDIRECT (src_id) | 86 | 22258 | | WithBestIndex | 23766 |
| TEMP_GM_TABLE_UNDIRECT (dst_id) | 166 | 22398 | | | |
| | | | | | |
| email-Enron.ungraph.txt | Create Time | Whole Time | | | Whole Time |
| GM_TABLE_UNDIRECTED (dst_id) | 613 | 23353 | | NoIndex | 24937 |
| TEMP_GM_TABLE_UNDIRECT (src_id) | 34 | 27131 | | WithBestIndex | 28229 |
| TEMP_GM_TABLE_UNDIRECT (dst_id) | 619 | 26148 | | | |

Figure 12: Creating Index Experiments on K-core Algorithm

| PageRank | | | | | |
|---|---|---|---|---|---|
| as-skitter.ungraph-75000.txt | Create Time | Whole Time | | | Whole Time |
| GM_TABLE (src_id) | 912 | 9834 | ✔ | NoIndex | 10478.61 |
| norm_table (src_id) | 693 | 9113 | ✔ | WithBestIndex | 8821 |
| offset_table (node_id) | 154 | 7273 | ✔ | | |
| GM_PAGERANK (node_id) | 269 | 9627 | ✔ | | |
| | | | | | |
| ca-AstroPh.txt | Create Time | Whole Time | | | Whole Time |
| GM_TABLE (src_id) | 1373 | 8312 | ✔ | NoIndex | 8086.47 |
| norm_table (src_id) | 844 | 6294 | ✔ | WithBestIndex | 9089 |
| offset_table (node_id) | 43 | 6640 | ✔ | | |
| GM_PAGERANK (node_id) | 47 | 5509 | ✔ | | |
| | | | | | |
| cit-HepPh.txt | Create Time | Whole Time | | | Whole Time |
| GM_TABLE (src_id) | 423 | 7254 | ✔ | NoIndex | 9045.31 |
| norm_table (src_id) | 1124 | 8086 | ✔ | WithBestIndex | 9394 |
| offset_table (node_id) | 67 | 6647 | ✔ | | |
| GM_PAGERANK (node_id) | 67 | 9074 | ✔ | | |
| | | | | | |
| cit-HepTh.txt | Create Time | Whole Time | | | Whole Time |
| GM_TABLE (src_id) | 1086 | 8797 | ✔ | NoIndex | 5930 |
| norm_table (src_id) | 1332 | 9011 | ✔ | WithBestIndex | 8067 |
| offset_table (node_id) | 52 | 6467 | ✔ | | |
| GM_PAGERANK (node_id) | 57 | 6061 | ✔ | | |
| | | | | | |
| com-amazon.ungraph-75000.txt | Create Time | Whole Time | | | Whole Time |
| GM_TABLE (src_id) | 608 | 5237 | ✔ | NoIndex | 5431 |
| norm_table (src_id) | 734 | 5291 | ✔ | WithBestIndex | 6672 |
| offset_table (node_id) | 167 | 4906 | ✔ | | |
| GM_PAGERANK (node_id) | 119 | 5313 | ✔ | | |
| | | | | | |
| com-dblp.ungraph-75000.txt | Create Time | Whole Time | | | Whole Time |
| GM_TABLE (src_id) | 290 | 5272 | ✔ | NoIndex | 5244 |
| norm_table (src_id) | 933 | 5773 | ✔ | WithBestIndex | 6750 |
| offset_table (node_id) | 178 | 4774 | ✔ | | |
| GM_PAGERANK (node_id) | 203 | 6694 | ✔ | | |
| | | | | | |
| email-Enron.ungraph.txt | Create Time | Whole Time | | | Whole Time |
| GM_TABLE (src_id) | 869 | 18740 | ✔ | NoIndex | 20539 |
| norm_table (src_id) | 1626 | 20199 | ✔ | WithBestIndex | 19712 |
| offset_table (node_id) | 91 | 16475 | ✔ | | |
| GM_PAGERANK (node_id) | 87 | 19306 | ✔ | | |
| | | | | | |
| email-EuAll.txt | Create Time | Whole Time | | | Whole Time |
| GM_TABLE (src_id) | 290 | 64077 | ✔ | NoIndex | 61180 |
| norm_table (src_id) | 1375 | 65333 | ✔ | WithBestIndex | 59470 |
| offset_table (node_id) | 682 | 55714 | ✔ | | |
| GM_PAGERANK (node_id) | 551 | 66967 | ✔ | | |

Figure 13: Creating Index Experiments on PageRank Algorithm

The following are the results of the degree distribution and pagerank results for the ten datasets.

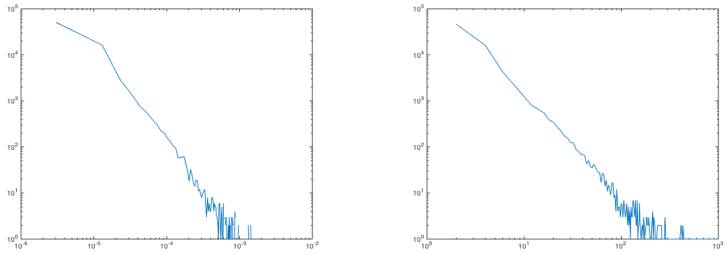


Figure 14: Result on PageRank Algorithm and Degree Distribution

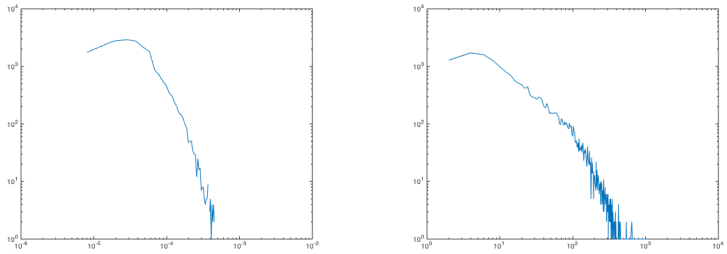


Figure 15: Result on PageRank Algorithm and Degree Distribution

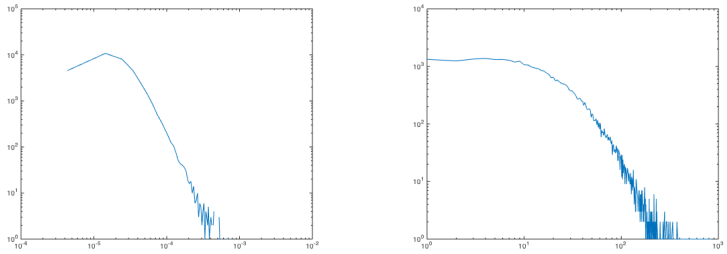


Figure 16: Result on PageRank Algorithm and Degree Distribution

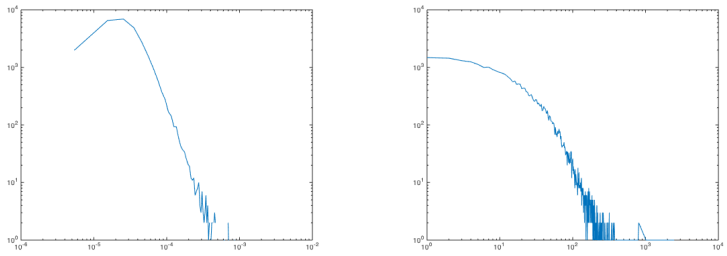


Figure 17: Result on PageRank Algorithm and Degree Distribution

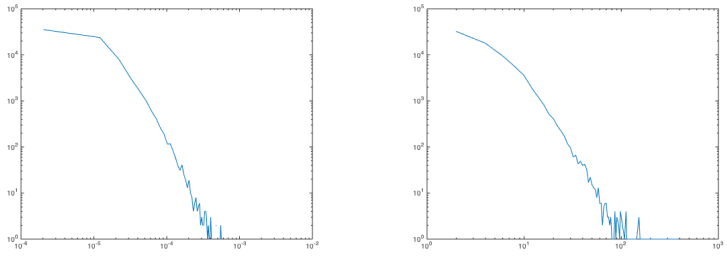


Figure 18: Result on PageRank Algorithm and Degree Distribution

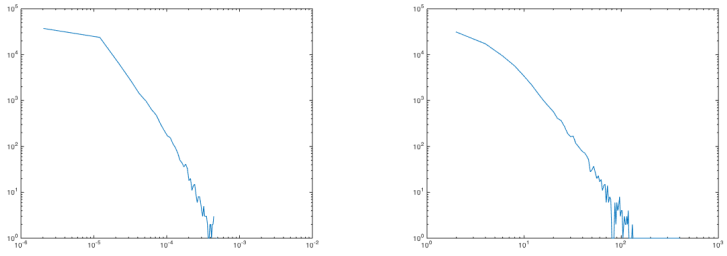


Figure 19: Result on PageRank Algorithm and Degree Distribution

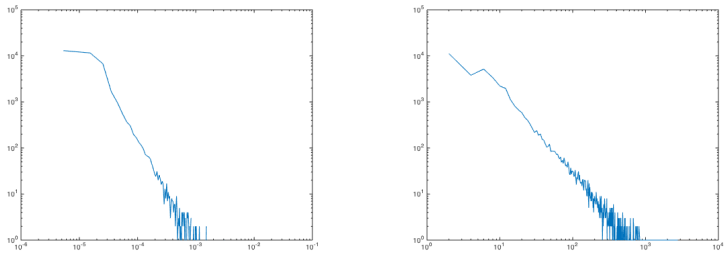


Figure 20: Result on PageRank Algorithm and Degree Distribution

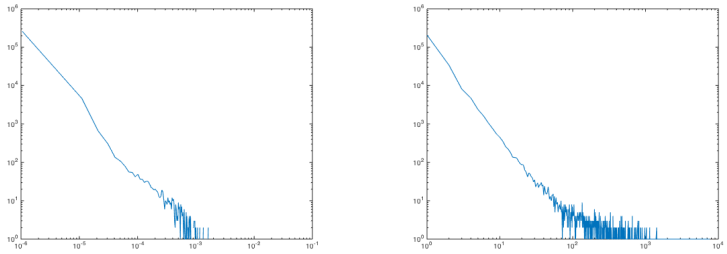


Figure 21: Result on PageRank Algorithm and Degree Distribution

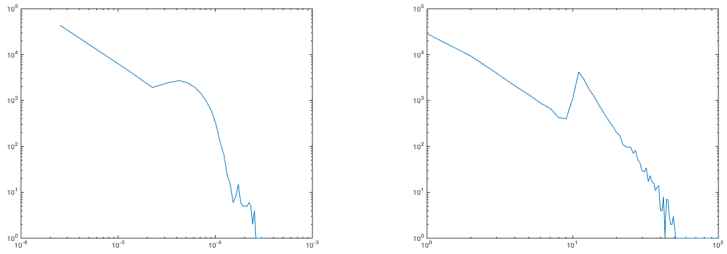


Figure 22: Result on PageRank Algorithm and Degree Distribution

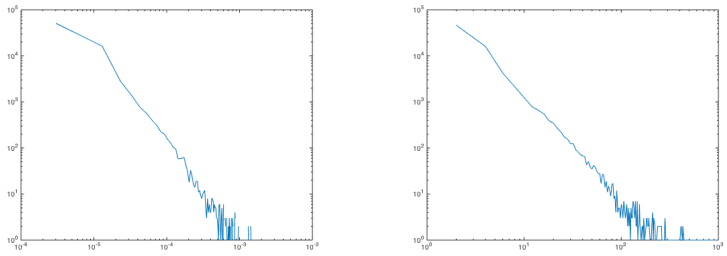


Figure 23: Result on PageRank Algorithm and Degree Distribution

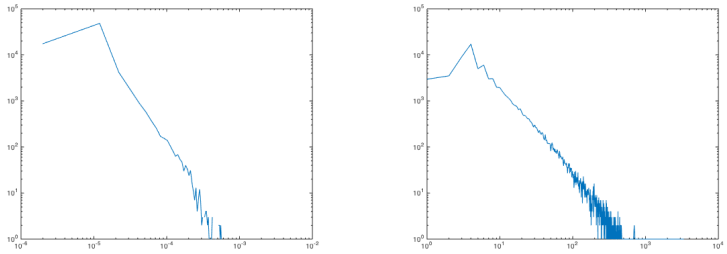


Figure 24: Result on PageRank Algorithm and Degree Distribution

## 4.3 Phase 3 Experiments on Real World Datasets

### 4.3.1 K-core

We also did more comprehensive experiments on the 20 datasets to test the functionality of algorithms in the framework and the K-core algorithm we implemented. Different datasets show different properties when graph mining algorithms are performed on them. We used K = 5 in the experiment as stated in the requirement of the project.

We can find in the table below that some datasets such as **as-Caida, Cit-HepPh, ca-AstroPh, cit-HepTh, soc-digg, soc-hamsterster, soc-Slashdot0811, text-spanishbook** have relatively big number of nodes but only one or a few connected components. Other datasets such as **as-skitter, com-amazon, com-dblp, email-Enron** have big number of nodes and relatively big number of connected components. One anomaly type is **soc-pokec, soc-flickr** in which both number of nodes and number of connected components equal 0. That suggests that no nodes have 5 cores requirement in the dataset. And the small number of connected components suggests that most of the nodes are connected in the graph, only very small number of clusters are formed. The explanation of the formation of connected components is that citations or user relationship form certain groups based on some properties such as interests, regions or languanges. For example, DBLP dataset has 764 connected components which suggest that many research field are relatively independent and do not cite

other areas of research much, resulting in multiple connected components formed. Also, the number of nodes of DBLP is relatively large compared to the original number of nodes, whici is , so we find infer that inside a research area, people cite each other frequently, which helps produce a large number of nodes. For datasets such as **soc-hamsterster** and **cit-HepTh** which include citation information in arXiv's High Energy Physics area and friendships between users of the website hamsterster.com, we can find that only one connected components are formed. The reason is that this is a small area of research or common interest and people in this group are well connected and no significant sub-clusters are formed. The large number of nodes compared to the total number of nodes suggest that research work and users in these fields are highly connected and active, or many of the nodes are of high quality.

For datasets such as **soc-pokec, soc-flickr** and **p2p-Gnutella** which no or very few connected components are formed and that the number of nodes in the K-core results are small. The explanation for few connected components are similar to the above, which means that relations of points in the dataset are already highly connected and form a group that no further significant sub-clusters form. The small number of nodes means that nodes such as users in digg.com and flickr.com do not have authorities or hubs who have a lot of followers. Instead, users have simple relationship with each other and are generally equal. Then K=5 K-core algorithm will generate very small number of nodes who have the coreness.

Table of K-core Algorithm Result for 20 Datasets

| Dataset | Number of Nodes | Number of Connected Components |
|---|---|---|
| as-Caida | 1436 | 1 |
| as-skitter | 9090 | 159 |
| bio-protein | 57 | 1 |
| ca-AstroPh | 12574 | 8 |
| cit-Cora | 384 | 13 |
| cit-HepPh | 8393 | 3 |
| cit-HepTh | 21768 | 2 |
| com-amazon | 19029 | 2090 |
| com-dblp | 23194 | 764 |
| email-Enron | 21577 | 180 |
| p2p-Gnutella31 | 88 | 7 |
| soc-digg | 1066 | 2 |
| soc-flickr | 0 | 0 |
| soc-hamsterster | 1103 | 1 |
| soc-pokec | 0 | 0 |
| soc-Slashdot0811 | 20577 | 2 |
| soc-Youtube | 277 | 4 |
| soft-jdkdependency | 84 | 1 |
| text-spanishbook | 1350 | 1 |

### 4.3.2 Degree Distribution

Degree Distribution Experiment

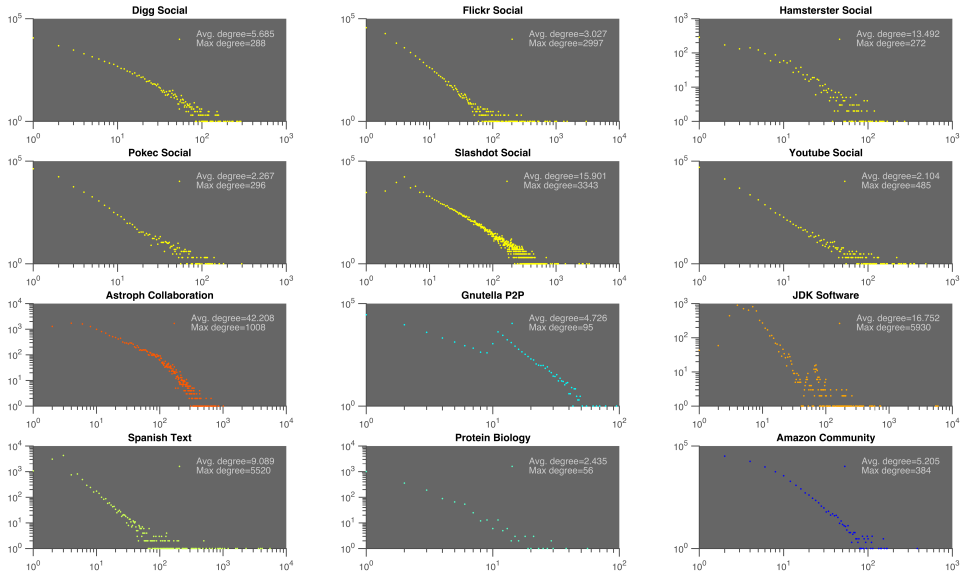



Figure 25: Result on Degree Distribution Experiment on 20 datasets:Degree

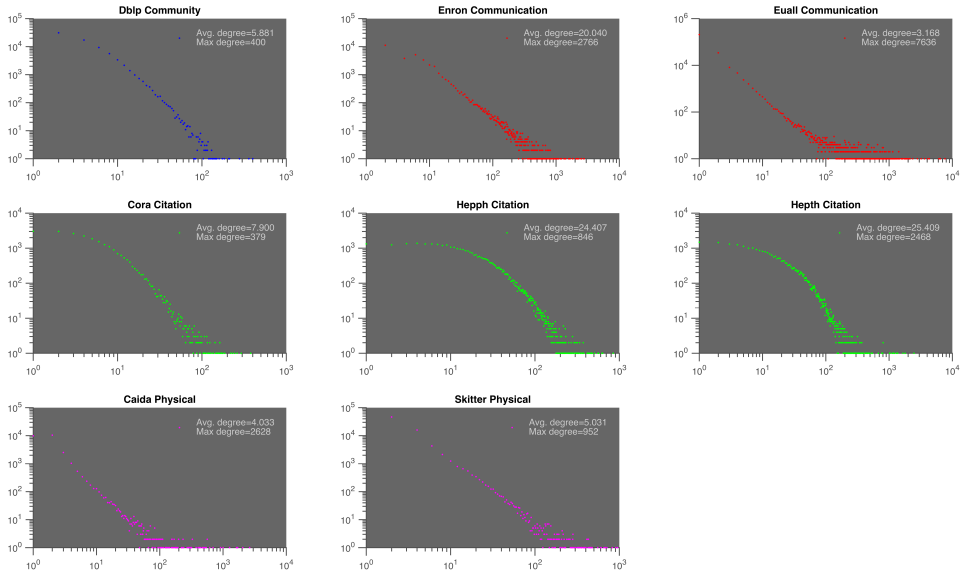



Figure 26: Result on Degree Distribution Experiment on 20 datasets:Degree

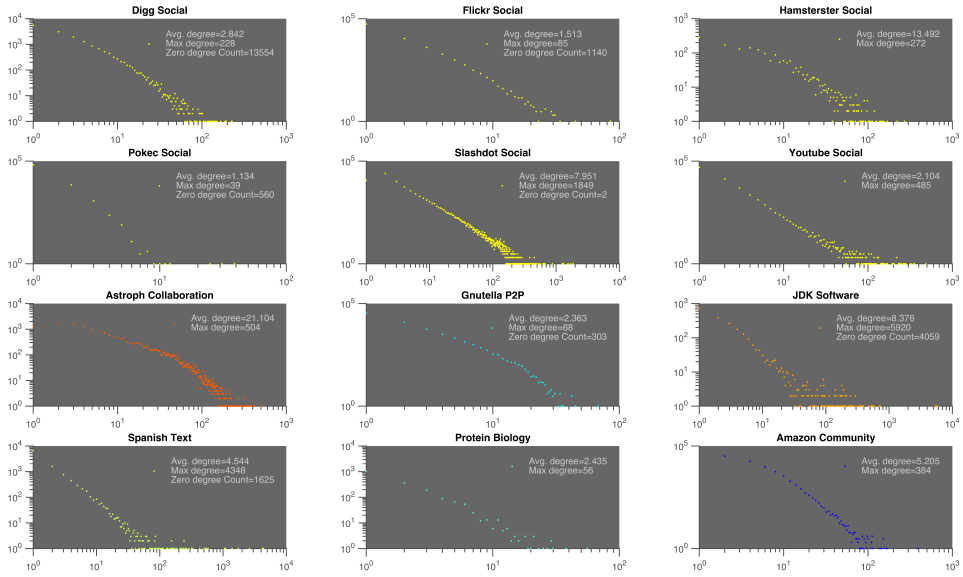



Figure 27: Result on Degree Distribution Experiment on 20 datasets:InDegree

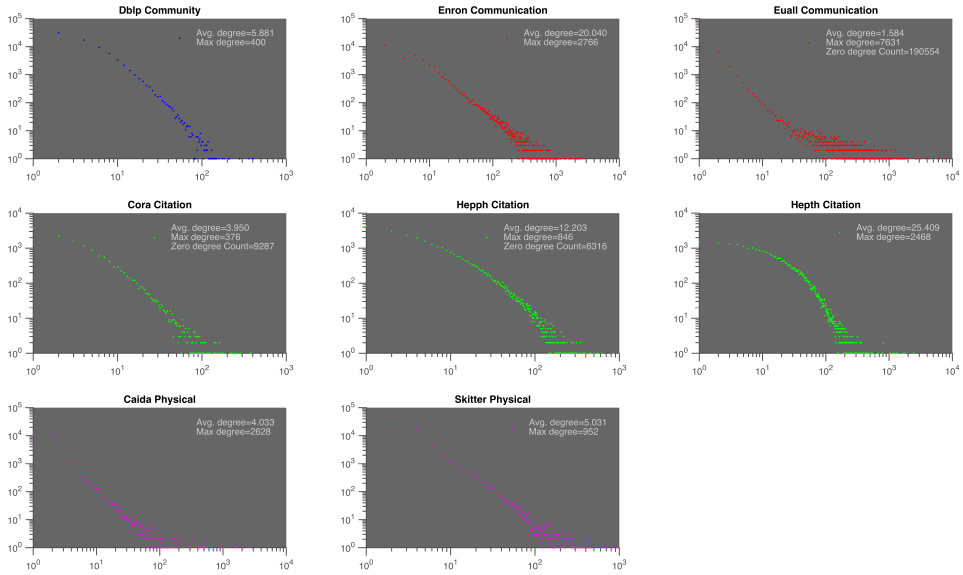



Figure 28: Result on Degree Distribution Experiment on 20 datasets:inDegree

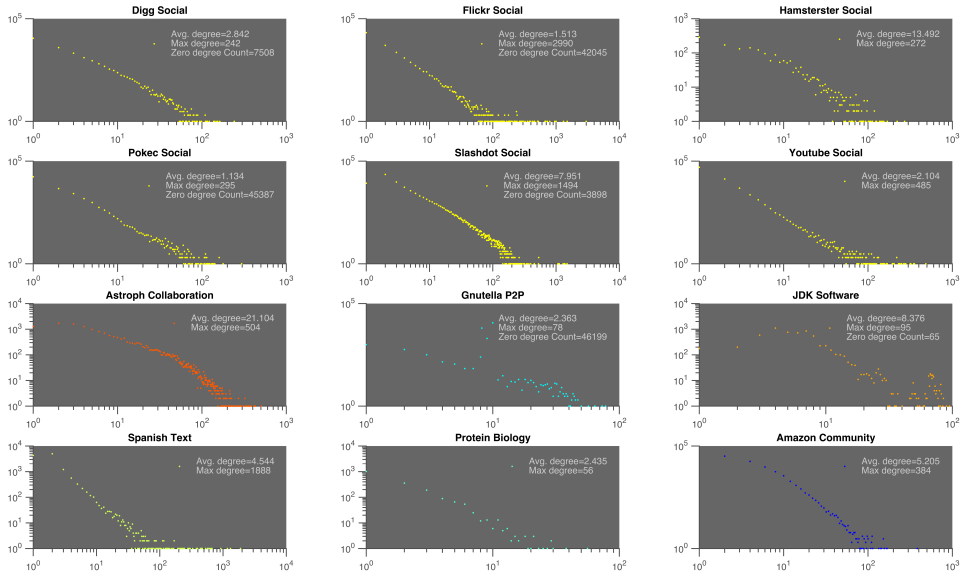



Figure 29: Result on Degree Distribution Experiment on 20 datasets:outDegree

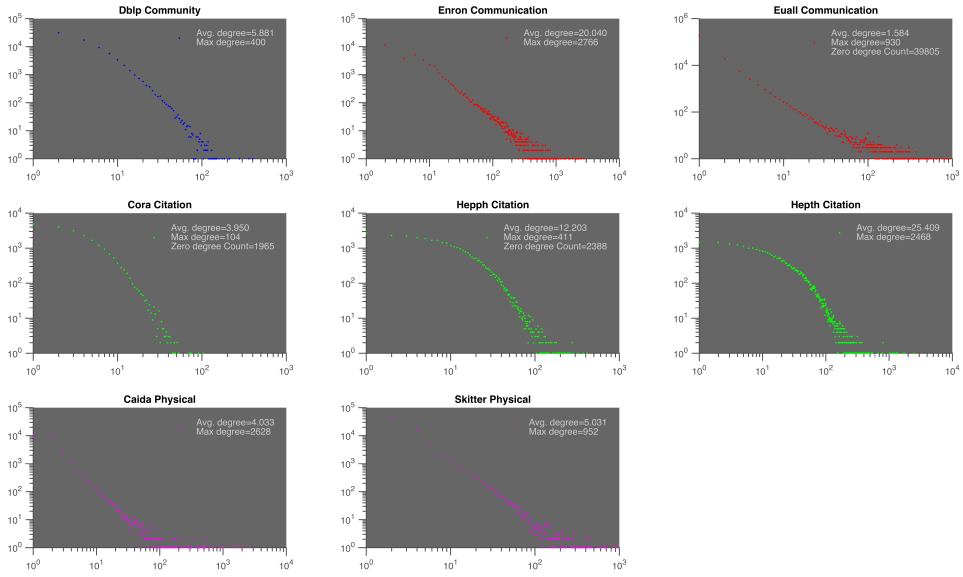



Figure 30: Result on Degree Distribution Experiment on 20 datasets:outDegree

Most of the graphs follow the power law. But for some of the graphs, we see strange

patterns. We discuss these below. 1. Gnutella P2P: The Out-degree distribution is not a power-law pattern and it has sudden jump in the middle of the picture. It means that about 10 machines connects to all of the machines in the p2p cluster. It might be some master file service machines that stores all of the files in the cluster.

2. JDK dependency: The Out-degree distribution is not a power law pattern, and In-degree distribution shows lots of waves at the tail. We inferred that in Java the relationship between classes is not just simple independent as Java has inherit, interface, implementation. For all the classes, the basement class is Object.class.

3. Flickr Social: From the Out-degree we found about more than 40000 users who has zero out degree, but from In-degree we found that only 1000 users has zero in degree. This shows that in Flickr most users just upload their photos but they don't care others so they don't follow others. And there are some users who follows large amount of users. Most of the users are followed by some maybe administrative account as most of the users have at least one follower.

4. Pokec Social: The same observation with Flickr also takes place on Pokec.

5. Email EuAll: From the in degree we found that there are almost 200000 accounts that receives zero email, which shows a large amount of mistyped, no-existing or spam email addresses.

### 4.3.3 PageRank

PageRank Experiment

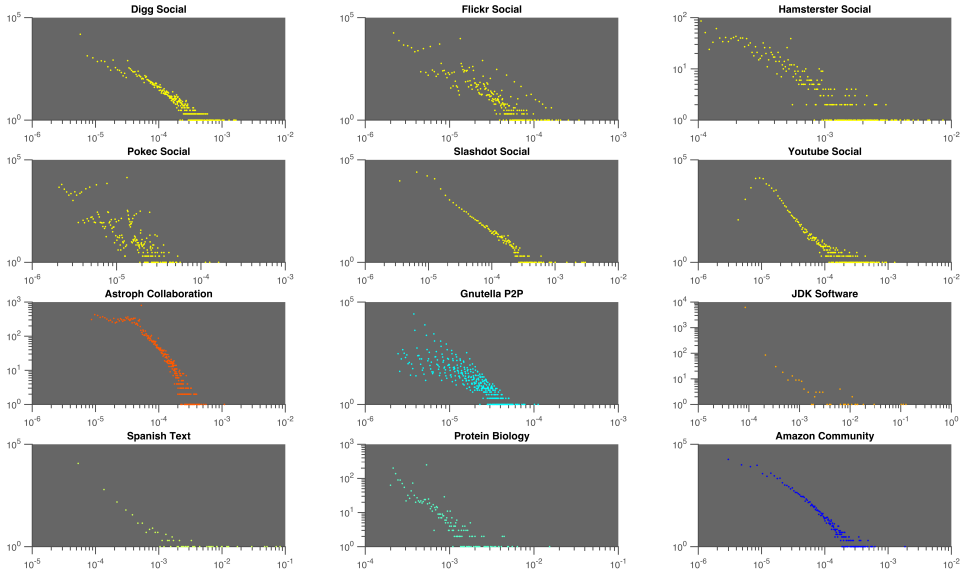



Figure 31: Result on PageRank Experiment on 20 datasets

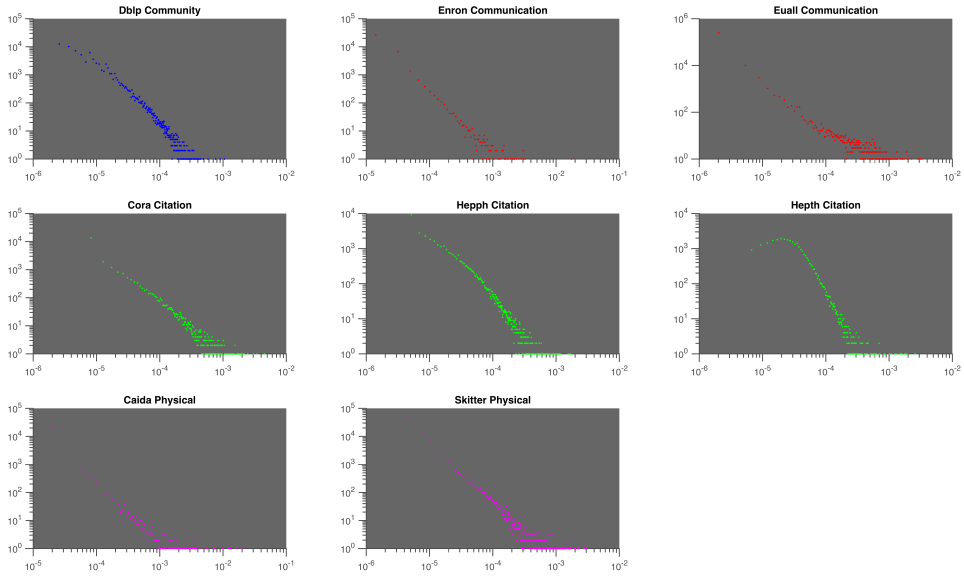



Figure 32: Result on PageRank Experiment on 20 datasets

We analyzed the pageRank score and the number of nodes with that score.Half of the graphs show the pattern of power-law, but others are not which described as below. 1. Digg Social: There are several small jump in the descending trend.

2. Flickr Social: the points are more scattered and distributed sparse compared with power-law. As we have mentioned above in degree distribution, due to the social relationship that most of the users are not following anyone else. Or those accounts are zombies accounts.

3. Hamsterster Social: The trending is more or less like power-law but it is not a standard one.

4. Pokec Social: Like Flicker, the points are more scattered. due to the social relationship that most of the users are not following anyone else. Or those accounts are zombies accounts.

5. Youtube Social: After a few ascending, it follows the power-law. We inferred that the reason is most of the Youtebe users using Google account to log in and do the social work. As not all of the google account are active users, there are small amount of very inactive user account which combines the start of the line.

6. Gnutella P2P: As in a P2P network, every node is totally equal to others, there is no way to say which one is more important or not. But there are pagerank difference is just because maybe some of the users has more files than others, but if they are normal users, no one has extraordinary numbers of files unless it has some purpose.

7. JDK dependency: As described in degree distribution, JDK classes has its internal dependency following Obejct Oriented Language Design.

8. Hepth Citation: There is a small ascending trend and then goes with power-law.

### 4.3.4 Connected Components

Connected Component Experiment

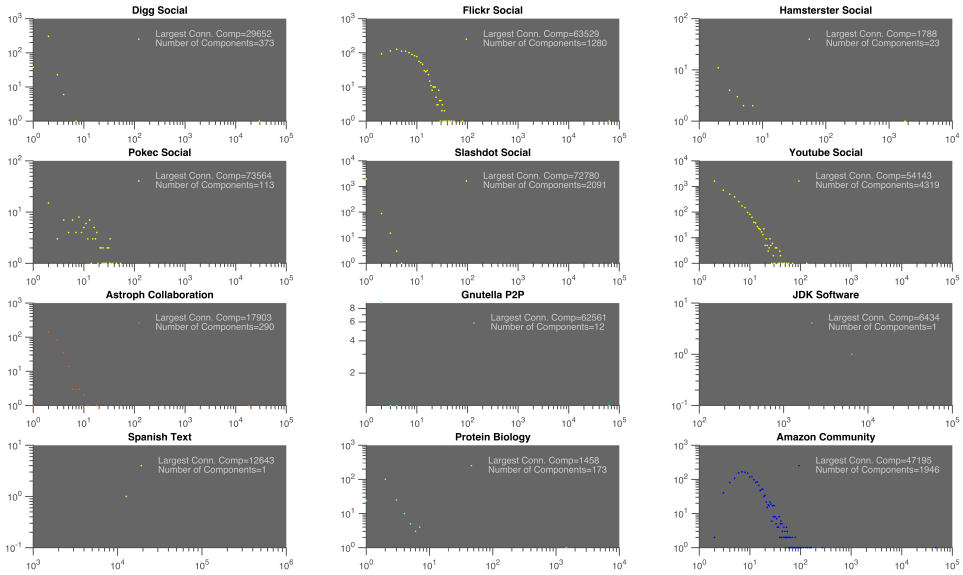



Figure 33: Result on Connected Component Experiment on 20 datasets

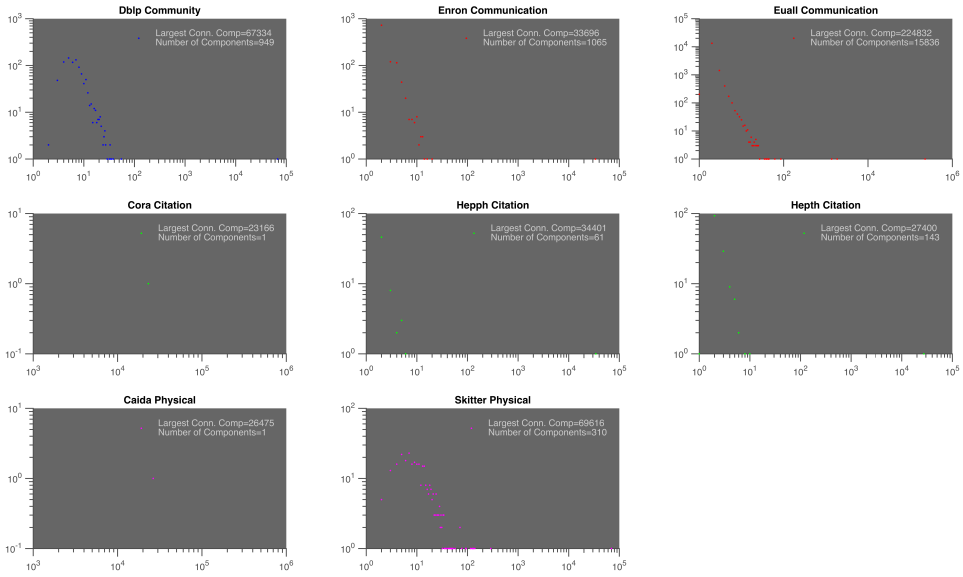



Figure 34: Result on Connected Component Experiment on 20 datasets

We analyze the component size and the number of components. We can observe a power relation for components if there are hundreds of components. In all the graphs, we see that either the graph is fully connected, or it has too little number of components or it follows the power-law. 1. Fully connected graph: JDK dependency is fully connected because it needs to follow JAVA rules.

Spanish Text is fully connected because it is a graph from words of a book. And words in the book are continuous, which means that all the words are of course in the place one by one and must be fully connected.

Caida AS is fully connected because all the ISP must be connected to the internet network, and make sure all their customers can reach out to where ever they want in the universe, so of course they are fully connected.

2. Few Components: If a graph shows there are few components, it shows that there are strong connection relationship between nodes. Like in Digg, Hamsterster, Pokec, their users are strongly connected to each other. And for Gnutalla P2P, the number of components is just 12. In protein biology graph, it shows that the protein structure is complicated as they highly connect to each other. And in Hepph and HepTh citation, as they are focusing on papers on High Energy Physics which is really high technical, there are not so many papers like some other fields, so the paper citation is limited.

### 4.3.5 Eigenvalue Experiments

Eigenvalues Experiment

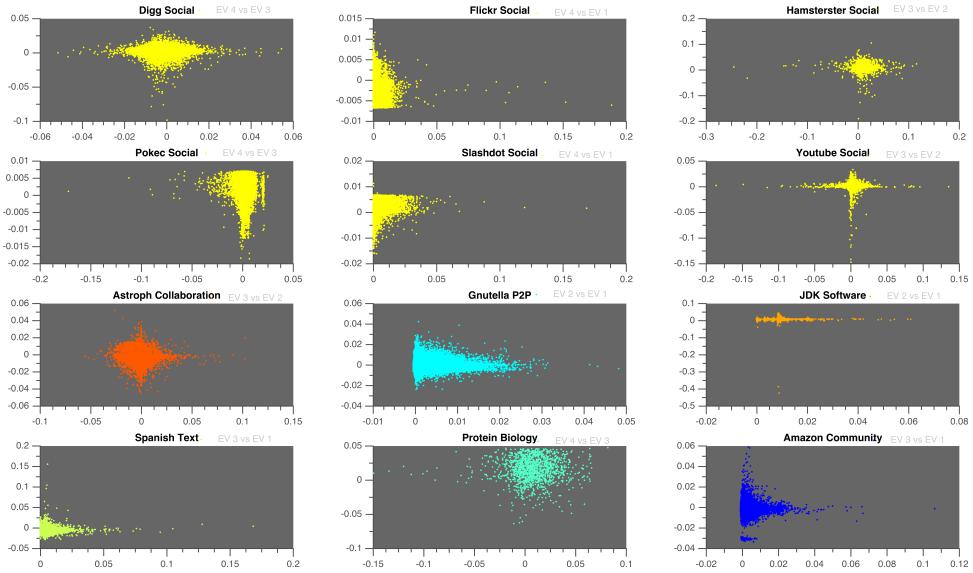



Figure 35: Result on Eigenvalue Experiment on 20 datasets

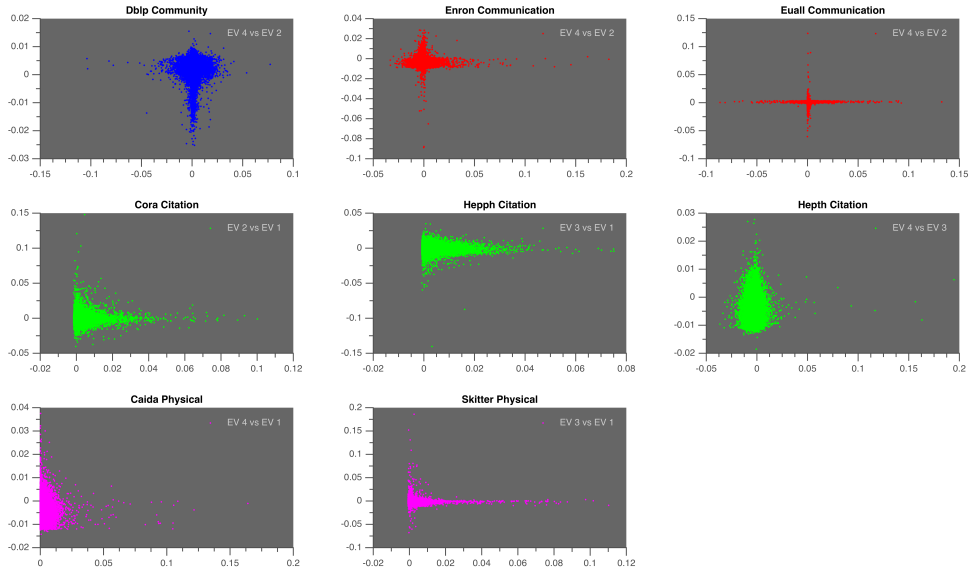



Figure 36: Result on Eigenvalue Experiment on 20 datasets

We performed eigenvalue calculation on the adjacency matrix of undirected graphs as

directed graphs' adjacency matrix is not symmetric and eigenvalues of such graphs can be complex numbers. Eigenvalue analysis is fundamental in terms of the spectral properties of graphs. It can also be used to find approximations to graph measures such as triangle counts. The algorithms implemented in the *graphMiner* framework that we used to do the experiments are Lanczos-SO and QR algorithms.

We plotted comparisons of eigen vectors for certain two of top three eigenvalues. Some datasets such as **soc-Youtube, Euall** show clear "spoke" like features. Datasets such as **Gnutella31, bio-protein** have no clear "spoke" features. In fact, if we only plot the very top nodes with the largest eigenvalues, the plots will show clearer spokes, for example, if we use top 100 nodes in the datasets with largest eigenvalues. The plots we have used are called EE-plots, showing the values of nodes in the $i_th$ eigenvector vs. in the $j_th$ eigenvector. Notice the clear spokes in top eigenvectors signify the existence of a strongly related group of nodes in near-cliques or bi-cliques.

With the idea of cliques in mind, we can infer in the datasets that datasets with clear spokes suggest that such datasets have clique relationship, where certain nodes in part of the graph are heavily connected to each other and thus form a small sub-group. **Euall** for example, is the sender and the recipient of the email data from a large European research institution in a period of time. Thus, people of aquaintance will tend to contact each other often, which create cliques. Similar cliques occur for **soc-Youtube** which contain users' friendship. The clique might come from people like the same channel or have similar subscriptions.

Other datasets do not show "spokes" as clear as the above datasets, but that might come from too many nodes are included. If we only plot part of the nodes in the dataset, we might find clearer spokes. Also, in such datasets for example **bio-protein, Hepth Citation**, we can still find that points in the EE-graph form a cluster instead of randomly distribute. Thus we can infer that there are still cliques in the data, but they are loosely close, that is, nodes in cliques also connect to other nodes that are not apparently similar to nodes in the clique.

#### 4.3.6 Triangle Count

Triangle Count Experiment

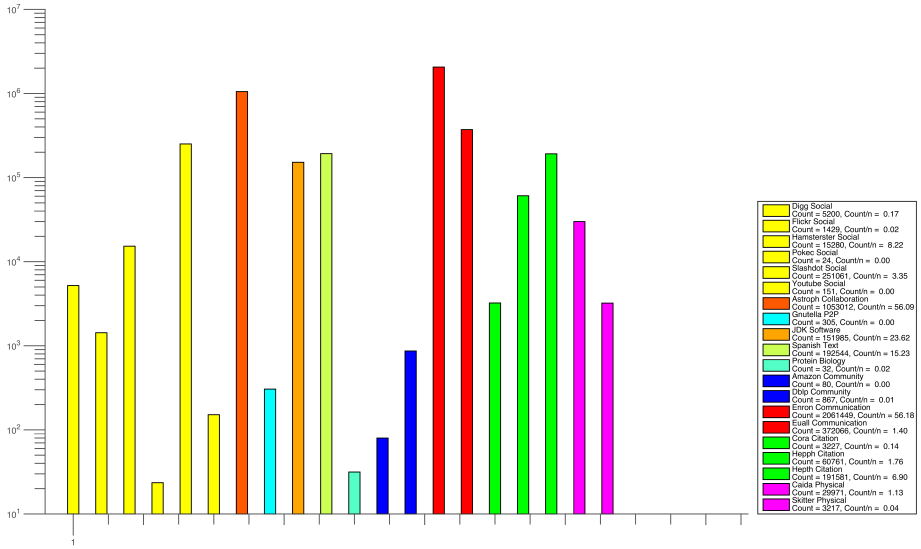



Figure 37: Result on Eigenvalue Experiment on 20 datasets

Triangle(three nodes connected to each other) count can answer questions about the community structure of graphs such as if nodes form starts or cliques. In the framework, approximation of triangle counts with high accuracy is done through its connection to eigenvalues. The total number of triangles in a graph is related to the sum of cubes of eigenvalues, and the first few eigenvalues provide extremely good approximations. A slightly more elaborate analysis approximates the number of triangles in which a node participates, using the cubes of the first few eigenvalues and the corresponding eigenvectors.

We can find in the graph that **Euron Email Communication** have the largest number of triangles and that **pokec-soc** have the smallest number of triangles. The email data contain emails of an organization for a period of time which apparently point to each other and explains why triangles form very often. In datasets where number of triangles are small, such as **pokec-soc**, that is because users friendship in this dataset seldom form triangles as the friendship is directed, which reduces the chance of forming a triangle.

# 5 Conclusions

Based on the experiments and the implementation of Kcore, we find that the Kcore algorithm can find subgraph structures effectively and we also find that the implemented algorithms in graphMiner, i.e. the pagerank, degree distribution, connected component etc works properly. Above this, we find that using SQL to do basic and advanced graph mining queries are viable and actually effective. We will have future exploration into this to make better use of SQL and advantage of database.

# References


[1] U Kang , *PEGASUS: Mining Peta-Scale Graphs*, 3rd ed. ICDM, 2009.

[2] U Kang , *GBASE: an efficient analysis platform for large graphs*, 3rd ed. VLDB, 2012.

[3] U Kang , *HADI: Mining Radii of Large Graphs*, 3rd ed. ACM Transactions on Knowledge Discovery from Data, 2010.

[4] Danai Koutra , *Unifying Guilt-by-Association Approaches: Theorems and Fast Algorithms*, 3rd ed. ECML PKDD, 2011.

[5] Danai Koutra , *k-core decomposition: a tool for the visualization of large scale networks*, 3rd ed. Advances in Neural Information Processing Systems, 2005.

# A Appendix

## A.1 Labor Division

The team performed the following tasks

- Implementation of Kcore [Jin Hu]
- Unit Tests and visualization of results [Ye Zhou]
- General debugging and testing[Ye Zhou and Jin Hu]
- Experiments and Analysis [Ye Zhou and Jin Hu]

## A.2 Acknowledgement

Thanks to Professor Christos Faloutsos for the wonderful lectures on various aspects of multimedia database and data mining topics. Thanks to TA Neil Shah for the careful design of the project and helpful feedback after each phase was completed. Thanks to Nijith Jacob and Sharif Doghmi, who took this class in a previous year for the *graphMiner* framework.

## A.3 Dataset and Origin

The realworld datasets are from SNAP, KONECT websites.

as-Caida.undir.txt-http://snap.stanford.edu/data/as-caida.html
bio-protein-undir.txt-http://konect.uni-koblenz.de/networks/moreno_propro
cit-Cora.txt-http://konect.uni-koblenz.de/networks/subelj_cora
soc-digg.txt-http://konect.uni-koblenz.de/networks/munmun_digg_reply
soc-flickr-75000.txt-http://konect.uni-koblenz.de/networks/flickr-growth
soc-hamsterster.undir.txt-http://konect.uni-koblenz.de/networks/petster-friendships-h
soc-pokec-75000.txt-http://snap.stanford.edu/data/soc-pokec.html
soc-Youtube-75000.undir.txt-http://konect.uni-koblenz.de/networks/com-youtube
soft-jdkdependency.txt-http://konect.uni-koblenz.de/networks/subelj_jdk
text-spanishbook.txt-http://konect.uni-koblenz.de/networks/lasagne-spanishbook
as-skitter.ungraph-75000.txt-http://konect.uni-koblenz.de/networks/as-skitter
ca-AstroPh.txt-http://konect.uni-koblenz.de/networks/ca-AstroPh
cit-HepPh.txt-http://konect.uni-koblenz.de/networks/cit-HepPh
cit-HepTh.txt-http://konect.uni-koblenz.de/networks/cit-HepTh
com-amazon.ungraph-75000.txt-http://snap.stanford.edu/data/com-Amazon.html
com-dblp.ungraph-75000.txt-http://snap.stanford.edu/data/com-DBLP.html
email-Enron.ungraph.txt-http://snap.stanford.edu/data/email-Enron.html
email-EuAll.txt-http://snap.stanford.edu/data/email-EuAll.html
p2p-Gnutella31.txt-http://snap.stanford.edu/data/p2p-Gnutella31.html
soc-Slashdot0811-75000.txt-http://snap.stanford.edu/data/soc-Slashdot0811.
html

# Contents