

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

November 7, 2014

Abstract

In this project, we investigate the possibility of using only SQL to do graph manipulations on *graphMiner*. For the first stage, we implement kcore algorithms within *graphMiner* framework and have unit test on kcore and other already implemented algorithms in the system to gain better understanding of methods like kcore, PageRank, Eigenvalue computation. We also test the algorithms with wiki-Vote and soc-Epinions1 dataset from SNAP.

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 efficient in implementing and testing some of the algorithms without the worries like IO and file read and write, etc.

This is an important problem, because graph manipulation and mining has many real world applications and is a very important topic in research community. 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 larger dataset like wiki-Vote and soc-Epinion1 from SNAP.

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, implemented on the top of Hadoop platform. In the paper, first they try to find out the common operation, which is the matrix-vector multiplication, 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 success of Pegasus with large data graph mining 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 support 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 cache 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 specific fields improvement: Careful Algorithm Choice, selective parallelization, blocking and skewness exploiting. Finally it turned out that performance improved in both scalability 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 scheduling 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 Guilt-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 closely 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 implement 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 visualization 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 visualization 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 experimental knowledge. Self adjusting parameters might be a huge improvement and can be an interesting topic to follow.

...

3 Proposed Method

We implemented K-core using SQL following the Batagelj and Zaversnik k-core variation. 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.

...

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.

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.

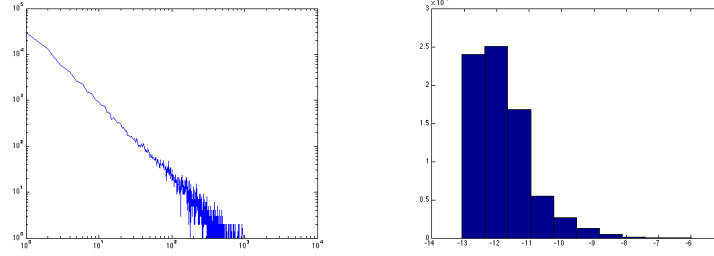


Figure 1: Degree Distribution(a) and PageRank(b) for Dataset SOC-Epinions1

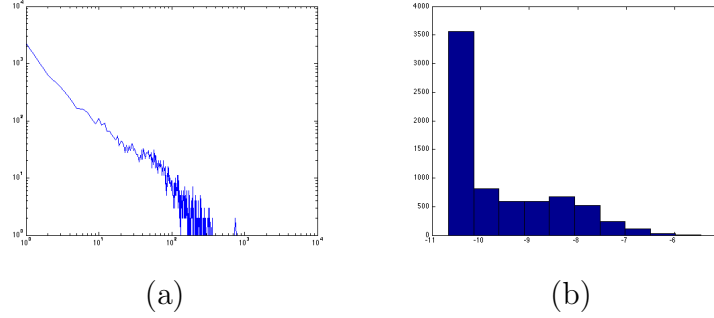


Figure 2: Degree Distribution(a) and PageRank(b) for Dataset wiki-Vote

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

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 unnecessary 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	Create Time	Whole Time		Combination Choice	Whole Time
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	✓	WithIndex	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	13 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.

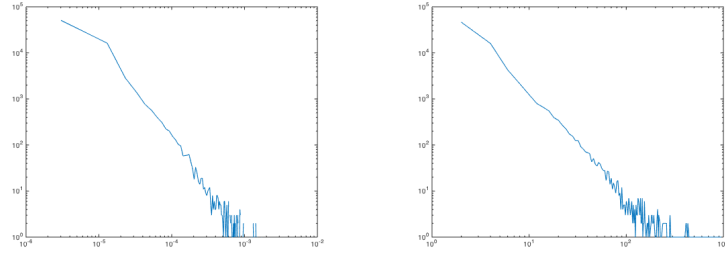


Figure 14: Result on PageRank Algorithm and Degree Distribution

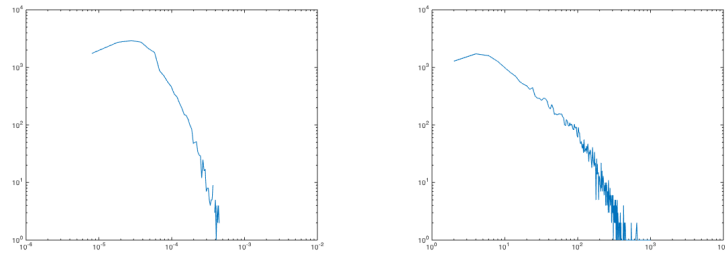


Figure 15: Result on PageRank Algorithm and Degree Distribution

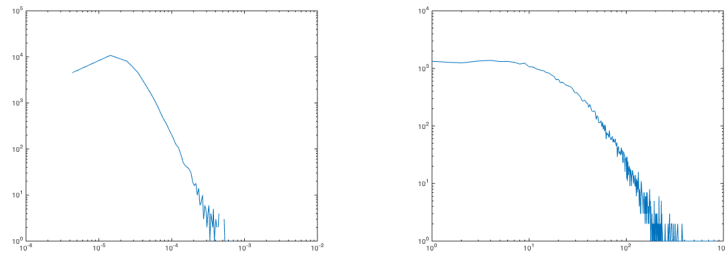


Figure 16: Result on PageRank Algorithm and Degree Distribution

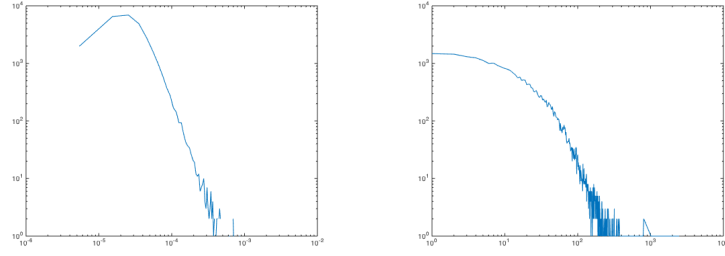


Figure 17: Result on PageRank Algorithm and Degree Distribution

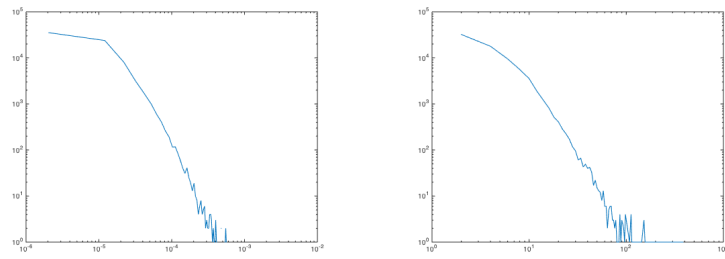


Figure 18: Result on PageRank Algorithm and Degree Distribution

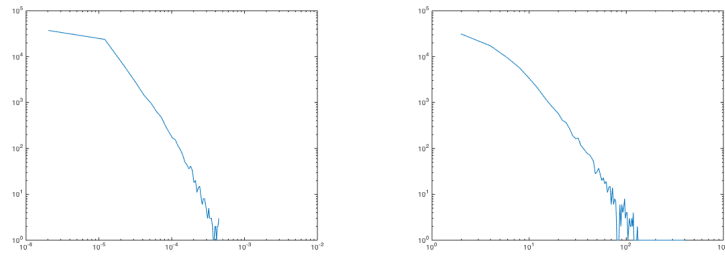


Figure 19: Result on PageRank Algorithm and Degree Distribution

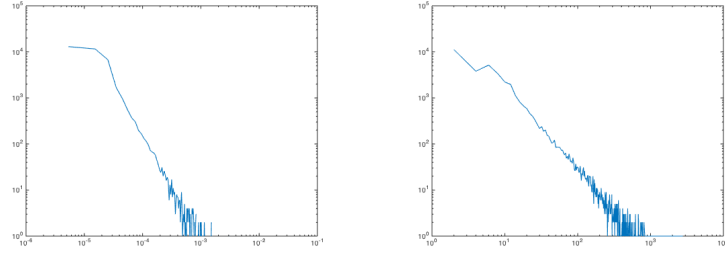


Figure 20: Result on PageRank Algorithm and Degree Distribution

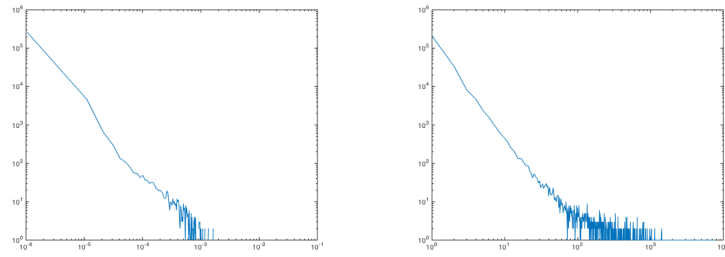


Figure 21: Result on PageRank Algorithm and Degree Distribution

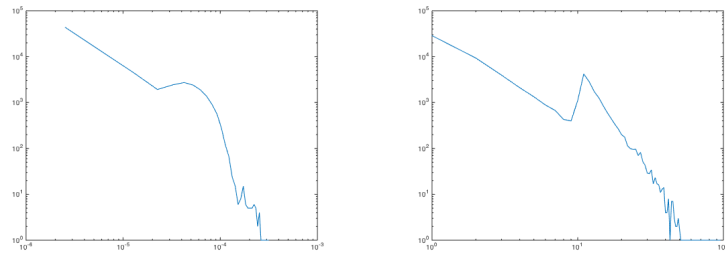


Figure 22: Result on PageRank Algorithm and Degree Distribution

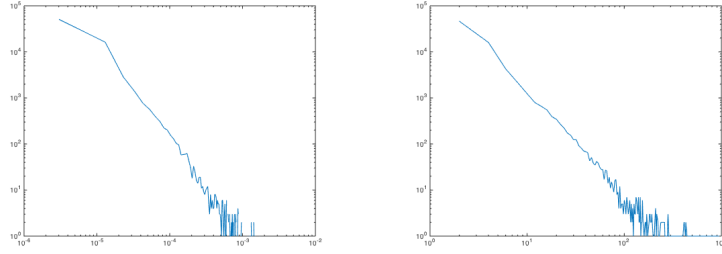


Figure 23: Result on PageRank Algorithm and Degree Distribution

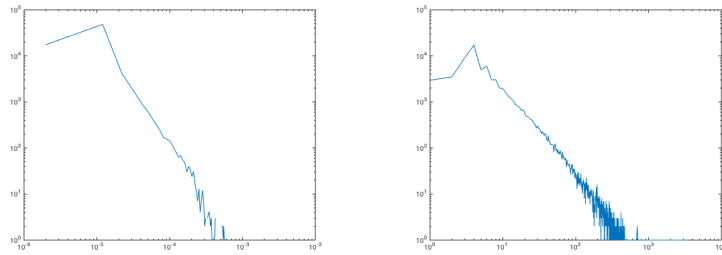


Figure 24: Result on PageRank Algorithm and Degree Distribution

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.

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]

A.2 Code for K-core

```
import argparse
```

```
from gm_params import *
from gm_sql import *
from math import sqrt
import os
import time
```

```
db_conn = None;
```

```
# Convert directed to undirected + remove multiple edges
```

```
def gm_to_undirected(rm_multiple = True):
```

```
    cur = db_conn.cursor()
```

```
    gm_sql_table_drop_create(db_conn, GM_TABLE_UNDIRECT, "src_id integer, dst_id integer, weight float")
```

```
    if rm_multiple:
```

```
        stmt = "INSERT INTO %s " % (GM_TABLE_UNDIRECT) + \
```

```
            " SELECT src_id, dst_id, AVG(weight) FROM " + \
```

```
            " (SELECT src_id, dst_id, weight FROM %s " % (GM_TABLE) + \
```

```
            " UNION ALL" + \
```

```
            " SELECT dst_id \"src_id\", src_id \"dst_id\", weight FROM %s " % (GM_TABLE) + \
```

```
            " GROUP BY src_id, dst_id"
```

```
    else:
```

```
        stmt = "INSERT INTO %s " % (GM_TABLE_UNDIRECT) + \
```

```
            " (SELECT src_id, dst_id, weight FROM %s " % (GM_TABLE) + \
```

```
            " UNION ALL" + \
```

```
            " SELECT dst_id \"src_id\", src_id \"dst_id\", weight FROM %s " % (GM_TABLE) + \
```

```

cur.execute(stmt)
db_conn.commit()

cur.close()

def gm_create_node_table ():
    cur = db_conn.cursor()

    gm_sql_table_drop_create(db_conn, GMNODES, "node_id integer")

    cur.execute ("INSERT INTO %s" % GMNODES +
                  " SELECT DISTINCT(src_id) FROM %s" % GM_TABLE_UN)

    db_conn.commit()

    cur.close()

def gm_save_tables (dest_dir, belief):
    print "Saving tables..."

    gm_sql_save_table_to_file(db_conn, GMDEGREE_DISTRIBUTION, "degree, count",
                              os.path.join(dest_dir, "degreedist.csv"), ",")
    gm_sql_save_table_to_file(db_conn, GMINDEGREE_DISTRIBUTION, "degree, count",
                              os.path.join(dest_dir, "indegredist.csv"), ",")
    gm_sql_save_table_to_file(db_conn, GMOUTDEGREE_DISTRIBUTION, "degree, count",
                              os.path.join(dest_dir, "outdegredist.csv"), ",")

    gm_sql_save_table_to_file(db_conn, GMNODE_DEGREES, "node_id, in_degree, out_degree",
                              os.path.join(dest_dir, "degree.csv"), ",");

    gm_sql_save_table_to_file(db_conn, GMPAGERANK, "node_id, page_rank", \
                              os.path.join(dest_dir, "pagerank.csv"), ",")

    gm_sql_save_table_to_file(db_conn, GMCONCOMP, "node_id, component_id",
                              os.path.join(dest_dir, "conncomp.csv"), ",")

    gm_sql_save_table_to_file(db_conn, GMRADIUS, "node_id, radius", \
                              os.path.join(dest_dir, "radius.csv"), ",");

    if (belief):
        gm_sql_save_table_to_file(db_conn, GMBELIEF, "node_id, belief", \

```

```

os.path.join(dest_dir,"belief.csv"), ",");

gm_sql_save_table_to_file(db_conn, GM_EIG_VALUES, "id, value", \
os.path.join(dest_dir,"eigval.csv"), ",");

gm_sql_save_table_to_file(db_conn, GM_EIG_VECTORS, "row_id, col_id, value", \
os.path.join(dest_dir,"eigvec.csv"), ",");

def kcore (args):
    global db_conn
    global GM_TABLE
    #Default Table names
    TEMP_GM_TABLE = "TEMP_GM_TABLE"
    TEMP_GM_TABLE_UNDIRECT = "TEMP_GM_TABLE_UNDIRECTED"
    TEMP_GM_NODES = "TEMP_GM_NODES"
    TEMP_GM_NODE_DEGREES = "TEMP_GM_NODE_DEGREES"
    REMOVED_NODE_TABLE = "REMOVED_NODE_TABLE"
    TEMP_GM_CON_COMP = "TEMP_GM_CON_COMP"
    temp_table = "GM_CC_TEMP"
    TEMP_GM_CON_COMP_2 = "TEMP_GM_CON_COMP_2"
    cur = db_conn.cursor()
    k = 5
    i = 2
    # create REMOVED_NODE_TABLE to record deleted node_id
    gm_sql_table_drop_create(db_conn, REMOVED_NODE_TABLE, "node_id integer")
    # create TEMP_GM_NODE_DEGREES
    gm_sql_table_drop_create(db_conn, TEMP_GM_NODE_DEGREES, "node_id integer, \
in_degree integer, out_degree integer")
    gm_sql_table_drop_create(db_conn, TEMP_GM_TABLE, "src_id integer, dst_id integer")
    cur.execute ("INSERT INTO %s" % TEMP_GM_TABLE + " SELECT src_id, dst_id, weight FROM %s" % GM_TABLE)
    cur.execute ("INSERT INTO %s" % TEMP_GM_NODE_DEGREES + " SELECT node_id, in_degree, out_degree FROM %s" % GM_TABLE)
    db_conn.commit()
    print "begin kcore iteration..."
    while(i<=5):
        cur.execute ("INSERT INTO %s" % REMOVED_NODE_TABLE +
                     " SELECT node_id FROM %s" % TEMP_GM_NODE_DEGREES + " WHERE in_degree < %d" % i)
        # remove degree < i nodes and associated edges in GM_TABLE and GM_TABLE_UNDIRECTED
        gm_sql_table_drop_create(db_conn, TEMP_GM_TABLE, "src_id integer, dst_id integer")
        gm_sql_table_drop_create(db_conn, TEMP_GM_TABLE_UNDIRECT, "src_id integer, dst_id integer")
        gm_sql_table_drop_create(db_conn, TEMP_GM_NODES, "node_id integer")
        cur.execute ("INSERT INTO %s" % TEMP_GM_NODES +

```

```

                                " SELECT DISTINCT(src_id) FROM %s" % GM_TABLE_UNDIRECT
cur.execute ("INSERT INTO %s" % TEMP_GM_TABLE + " SELECT src_id, dst_id,
cur.execute ("INSERT INTO %s" % TEMP_GM_TABLE_UNDIRECT + " SELECT src_id,
db_conn.commit()

# Recomputing node degrees...
gm_sql_table_drop_create(db_conn, TEMP_GM_NODE_DEGREES, "node_id integer,
                                in_degree integer, out_degree integer")
cur.execute ("INSERT INTO %s" % TEMP_GM_NODE_DEGREES +
                                " SELECT node_id, SUM(in_degree) \"in_degree\",
                                " (SELECT dst_id \"node_id\", count(*) \"in_deg
                                0 \"out_degree\" FROM %s" % TEMP_GM_TABLE +
                                " GROUP BY dst_id" +
                                " UNION ALL" +
                                " SELECT src_id \"node_id\", 0 \"in_degree\", \
                                count(*) \"out_degree\" FROM %s" % TEMP_GM_TAB
                                " GROUP BY src_id) \"TAB\" " +
                                " GROUP BY node_id")

db_conn.commit()
# print "iteration when i=", i
# gm_sql_print_table(db_conn, REMOVED_NODE_TABLE)
# print "TEMP_GM_NODES"
# gm_sql_print_table(db_conn, TEMP_GM_NODES)
# print "TEMP_GM_TABLE"
# gm_sql_print_table(db_conn, TEMP_GM_TABLE)
# print "TEMP_GM_NODE_DEGREES"
# gm_sql_print_table(db_conn, TEMP_GM_NODE_DEGREES)
i = i+1

# gm_sql_print_table(db_conn, TEMP_GM_NODES)
# print "finish kcore iteration, now calculate connected components"
# gm_sql_print_table(db_conn, TEMP_GM_NODE_DEGREES)

# Connected components
# Create CC table and initialize component id to node id

gm_sql_create_and_insert(db_conn, TEMP_GM_CON_COMP, GM_NODES, \
                                "node_id integer, component_id integer", \
                                "node_id, component_id", "node_id, node_id")
# gm_sql_print_table(db_conn, TEMP_GM_TABLE_UNDIRECT)
while True:
    gm_sql_table_drop_create(db_conn, temp_table, "node_id integer, componen

```

```

# Set component id as the min{component ids of neighbours, node's compo
cur.execute("INSERT INTO %s " % temp_table +
            " SELECT node_id, MIN(component_id) \"component_id\" \
            " SELECT src_id \"node_id\", MIN(component_id) \
            " WHERE dst_id = node_id GROUP BY src_id" +
            " UNION" +
            " SELECT * FROM %s" % TEMP_GMLCON_COMP +
            " ) \"T\" GROUP BY node_id")

db_conn.commit()
diff = gm_sql_vect_diff(db_conn, TEMP_GMLCON_COMP, temp_table, "node_id")
# Copy the new component ids to the component id table
gm_sql_create_and_insert(db_conn, TEMP_GMLCON_COMP, temp_table, \
                        "node_id integer, component_id integer", \
                        "node_id, component_id", "node_id, componen

print "Error = " + str(diff)
# Check whether the component ids has converged
if (diff == 0):
    print "Component IDs has converged"
    break

cur.execute ("SELECT count(distinct component_id) FROM %s" % TEMP_GMLCON_CO
num_components = cur.fetchone()[0]
print "Number of Components =", num_components
print "Now output decomposition (node_id,component_id) pairs"
cur.execute ("SELECT node_id, component_id FROM %s" % TEMP_GMLCON_COMP + "
for x in cur:
    print x
print "finished kcore, writing to files"
gm_sql_table_drop_create(db_conn, TEMP_GMLCON_COMP2,"node_id integer, comp
cur.execute ("INSERT INTO %s" % TEMP_GMLCON_COMP2 + " SELECT node_id, comp
gm_sql_save_table_to_file(db_conn, TEMP_GMLCON_COMP2, "node_id, component
                        os.path.join(args.dest_dir,"kcore_conncomp.cs

# Drop temp tables
gm_sql_table_drop(db_conn, temp_table)
gm_sql_table_drop(db_conn, TEMP_GMLCON_COMP)
cur.close()
return

```

#Project Tasks

#Task 1: Degree distribution

```

#-----
def gm_node_degrees ():

```



```

cur = db_conn.cursor()

# Create Table to store node degrees
# If the graph is undirected, all the degree values will be the same
print "Computing Node degrees..."

gm_sql_table_drop_create(db_conn, GM_NODE_DEGREES, "node_id integer, \
    in_degree integer, out_degree integer")

cur.execute ("INSERT INTO %s" % GM_NODE_DEGREES +
    " SELECT node_id, SUM(in_degree) \"in_degree\", \
    \" (SELECT dst_id \"node_id\", count(*) \"in_deg\
    0 \"out_degree\" FROM %s" % GM_TABLE +
    " GROUP BY dst_id" +
    " UNION ALL" +
    " SELECT src_id \"node_id\", 0 \"in_degree\", \
    count(*) \"out_degree\" FROM %s" % GM_TABLE +
    " GROUP BY src_id) \"TAB\" " +
    " GROUP BY node_id")

db_conn.commit()

cur.close()

# Degree distribution
def gm_degree_distribution (undirected):

    cur = db_conn.cursor()
    print "Computing Degree distribution of the nodes..."

    gm_sql_table_drop_create(db_conn, GM_DEGREE_DISTRIBUTION, "degree integer
    gm_sql_table_drop_create(db_conn, GM_INDEGREE_DISTRIBUTION, "degree integ
    gm_sql_table_drop_create(db_conn, GM_OUTDEGREE_DISTRIBUTION, "degree inte

    cur.execute ("INSERT INTO %s" % GM_INDEGREE_DISTRIBUTION +
        " SELECT in_degree \"degree\", count(*) FROM %s"
        " GROUP BY in_degree");

    cur.execute ("INSERT INTO %s" % GM_OUTDEGREE_DISTRIBUTION +
        " SELECT out_degree \"degree\", count(*) FROM %s"
        " GROUP BY out_degree");

```

```

if (undirected):
    # Degree distribution is same as in/out degree distribution for undir
    cur.execute ("INSERT INTO %s" % GM_DEGREE_DISTRIBUTION +
                  " SELECT * FROM %s" % GM_INDEGREE_DISTRIBUTION);
else:
    cur.execute ("INSERT INTO %s" % GM_DEGREE_DISTRIBUTION +
                  " SELECT in_degree+out_degree \"degree\", count(*)\n" +
                  " GROUP BY in_degree+out_degree");

db_conn.commit()
cur.close()

# Task 2: PageRank
# -----
def gm_pagerank (num_nodes, max_iterations = gm_param_pr_max_iter, \
                 stop_threshold = gm_param_pr_thres, damping_factor = gm_p

    offset_table = "GM_PR_OFFSET"
    next_table = "GM_PR_NEXT"
    norm_table = "GM_PR_NORM"

    cur = db_conn.cursor();
    print "Computing PageRanks..."

    gm_sql_table_drop_create(db_conn, norm_table, "src_id integer, dst_id inte

# Create normalized weighted table
cur.execute("INSERT INTO %s " % norm_table +
            " SELECT src_id, dst_id, weight/weight_sm \"weight\" FROM %s \"TA
            " (SELECT src_id \"node_id\", sum(weight) \"weight_sm\" FROM %s G
            " WHERE \"TAB1\".src_id = \"TAB2\".node_id")
db_conn.commit();

# Create PageRank Table and initialize to 1/n
gm_sql_create_and_insert(db_conn, GMPAGERANK, GM_NODES, \
                        "node_id integer, page_rank double precision def
                        "node_id", "node_id")

# Create offset table and initialize to 1-c/n
gm_sql_create_and_insert(db_conn, offset_table, GM_NODES, \
                        "node_id integer, page_rank double precision def
                        "node_id", "node_id")

```

```

num_iterations = 0
while True:
    # Create Table to store the next pageRank
    gm_sql_table_drop_create(db_conn, next_table, "node_id integer, page_r

    # Compute Next PageRank
    cur.execute ("INSERT INTO %s " % next_table +
                  " SELECT node_id, SUM(page_rank) FROM (" +
                  " SELECT dst_id \"node_id\", SUM(%s*weight*pa
                  " FROM %s, %s" % (norm_table, GMPAGERANK) +
                  " WHERE src_id = node_id GROUP BY dst_id" +
                  " UNION ALL" +
                  " SELECT node_id, page_rank * val \"page_rank
                  " FROM %s, (SELECT SUM(page_rank) \"val\" FRO
                  " ) \"TAB\" GROUP BY node_id" )

    db_conn.commit()

    diff = gm_sql_vect_diff(db_conn, GMPAGERANK, next_table, \
                            "node_id", "node_id", "page_rank", "page_rank

    # Copy the new page rank to the page rank table
    gm_sql_create_and_insert(db_conn, GMPAGERANK, next_table, \
                             "node_id integer, page_rank double precis
                             "node_id, page_rank", "node_id, page_rank

    num_iterations = num_iterations + 1
    print "Iteration = %d, Error = %f" % (num_iterations, diff)

    if (diff<=stop_threshold or num_iterations>=max_iterations):
        break

    # Drop temp tables
    gm_sql_table_drop(db_conn, offset_table)
    gm_sql_table_drop(db_conn, next_table)
    gm_sql_table_drop(db_conn, norm_table)

    cur.close()

# Task 3: Weakly Connected Components
#-----#
def gm_connected_components (num_nodes):

```

```

temp_table = "GMCC_TEMP"
cur = db_conn.cursor()
print 'Computing Weakly Connected Components...'

# Create CC table and initialize component id to node id
gm_sql_create_and_insert(db_conn, GMCON_COMP, GMNODES, \
                        "node_id integer, component_id integer", \
                        "node_id, component_id", "node_id, node_id")

while True:
    gm_sql_table_drop_create(db_conn, temp_table, "node_id integer, component_id integer")

    # Set component id as the min{component ids of neighbours, node's component id}
    cur.execute("INSERT INTO %s " % temp_table +
                " SELECT node_id, MIN(component_id) \"component_id\" "
                " SELECT src_id \"node_id\", MIN(component_id) \"component_id\" "
                " WHERE dst_id = node_id GROUP BY src_id" +
                " UNION" +
                " SELECT * FROM %s" % GMCON_COMP +
                " ) \"T\" GROUP BY node_id")

    db_conn.commit()

    diff = gm_sql_vect_diff(db_conn, GMCON_COMP, temp_table, "node_id", "component_id")

    # Copy the new component ids to the component id table
    gm_sql_create_and_insert(db_conn, GMCON_COMP, temp_table, \
                            "node_id integer, component_id integer", \
                            "node_id, component_id", "node_id, component_id")

    print "Error = " + str(diff)
    # Check whether the component ids has converged
    if (diff == 0):
        print "Component IDs has converged"
        break

    cur.execute ("SELECT count(distinct component_id) FROM %s" % GMCON_COMP)
    num_components = cur.fetchone()[0]

    print "Number of Components =", num_components
    cur.close()

# Drop temp tables

```

```

gm_sql_table_drop(db_conn, temp_table)

# Task 4: Radius of every node
#-----#
def gm_all_radius (num_nodes, max_iter = gm_param_radius_max_iter):

    hop_table = "GMRD_HOP"
    max_hop_ngh = "GMRD_MAX_HOP_NGH"

    cur = db_conn.cursor()
    print 'Computing radius of every node...'

    # initialize hop 0 table's hash
    gm_sql_create_and_insert(db_conn, hop_table+"0", GMNODES, \
                            "node_id integer, hash integer", \
                            "node_id, hash", "node_id, (((node_id%%s)+1)#(no

    for cur_hop in range(1,max_iter+1):
        print "Hop number : " + str(cur_hop)

        # create ith hop table
        cur_hop_table = hop_table+str(cur_hop)
        prev_hop_table = hop_table+str(cur_hop-1)
        gm_sql_table_drop_create(db_conn, cur_hop_table, "node_id integer, has
        cur.execute("INSERT INTO %s " % cur_hop_table +
                    " SELECT node_id, bit_or(hash) FROM ( " +
                    " SELECT src_id \"node_id\", bit_or(hash) \"hash\"
                    " FROM %s,%s" % (GM_TABLE_UNDIRECT, prev_hop_table)
                    " WHERE dst_id = node_id GROUP BY src_id " +
                    " UNION ALL" +
                    " SELECT * FROM %s ) \"TAB\" GROUP BY node_id" %

        db_conn.commit()

        # Check convergence
        diff = gm_sql_vect_diff(db_conn, cur_hop_table, prev_hop_table, "node

        print "Current Error = " + str(diff)
        if (diff==0):
            print "Convergence acheived"
            break

    nghbourhd_func = "2^( floor(log(2,hash)+1))/0.77351"

```

```

gm_sql_create_and_insert(db_conn, max_hop_ngh, cur_hop_table, \
                        "id integer, value double precision", \
                        "id, value", "node_id, %s" % (nghbourhd_func))

gm_sql_table_drop_create(db_conn, GM_RADIUS, "node_id integer, radius integer")

for i in range(0, cur_hop+1):
    print "Getting nodes with eff. radius " + str(i)
    # effective radius is the hop at which neighbour function value exceeds
    # 0.9 * the value at max hop
    cur.execute("INSERT INTO %s" % GM_RADIUS +
                " SELECT node_id, %s \"radius\" FROM %s, %s " % (i, h,
                " WHERE node_id = id AND %s >= 0.9*value " % (nghbourhd_func))

    db_conn.commit()
    cur.execute("DELETE FROM %s WHERE id IN (SELECT node_id FROM %s)" % (n,
    db_conn.commit()

cur.execute ("SELECT max(radius) FROM %s" % GM_RADIUS)
max_radius = cur.fetchone()[0]
print "Maximum effective radius =", max_radius

# drop temp tables
gm_sql_table_drop(db_conn, max_hop_ngh)
for i in range(0, cur_hop+1):
    gm_sql_table_drop(db_conn, hop_table+str(i))

cur.close()

# Task 5: Eigen values
# ----- #
# The adjacency matrix should be symmetric

def gm_eigen_QR_decompose(T, n, Q, R):
    G = "GM_QR_DECOMPOSE_GIVENS"
    temp_table = "GM_QR_DECOMPOSE_TEMP"
    I = "GM_QR_DECOMPOSE_IDENTITY"

```

```

cur = db_conn.cursor();

gm_sql_table_drop_create(db_conn, R,"row_id integer, col_id integer, value integer");

# Initialize R = T
cur.execute("INSERT INTO %s" % (R) + " SELECT * FROM %s" % (T))
db_conn.commit()

for i in range(1,n):
    # Compute the givens matrix
    cur.execute("SELECT value FROM %s " % (R) +
                "WHERE col_id = %s AND row_id >= %s ORDER BY row_id" % (i,i))

    c = cur.fetchone()[0]
    s = cur.fetchone()[0]
    r = sqrt(c*c + s*s)
    c = c/r
    s = -s/r

    gm_sql_table_drop_create(db_conn, G,"row_id integer, col_id integer, value integer");
    cur.execute("INSERT INTO %s" % (G) + " SELECT * FROM %s" % (I))
    cur.execute('UPDATE %s' % (G) + ' SET value = %s WHERE row_id = %s AND col_id = %s' % (i,i,i))
    cur.execute('UPDATE %s' % (G) + ' SET value = %s WHERE row_id = %s AND col_id = %s' % (i,i,i))
    cur.execute('INSERT INTO %s' % (G) + ' VALUES (%s,%s,%s)' % (str(i),str(i),str(i)))
    cur.execute('INSERT INTO %s' % (G) + ' VALUES (%s,%s,%s)' % (str(i+1),str(i+1),str(i+1)))
    db_conn.commit()

    # Compute Q
    if i == 1:
        # insert G*
        gm_sql_table_drop_create(db_conn, Q,"row_id integer, col_id integer, value integer");
        cur.execute("INSERT INTO %s" % (Q) + " SELECT \"col_id\" row_id, value row_id, row_id row_id, row_id row_id");
    else:
        gm_sql_table_drop_create(db_conn, temp_table,"row_id integer, col_id integer, value integer");
        gm_sql_mat_mat_multiply (db_conn, Q, G, temp_table, "col_id", "col_id", "value", "row_id", "row_id", "row_id");
        gm_sql_table_drop_create(db_conn, Q,"row_id integer, col_id integer, value integer");
        cur.execute("INSERT INTO %s" % (Q) + " SELECT * FROM %s" % (temp_table));

    db_conn.commit()
    # Compute R

```

```

gm_sql_table_drop_create(db_conn, temp_table,"row_id integer, col_id
gm_sql_mat_mat_multiply (db_conn, G, R, temp_table, "col_id", "row_id
                        "value", "row_id", "col_id", "ro
gm_sql_table_drop_create(db_conn, R,"row_id integer, col_id integer,
cur.execute("INSERT INTO %s" % (R) + " SELECT * FROM %s" % (temp_table)

db_conn.commit()

cur.close()
# Drop temp tables
gm_sql_table_drop(db_conn, G)

gm_sql_table_drop(db_conn, temp_table)

def gm_eigen_QR_iterate(T, n, EVal, EVec, steps, err):

    Q = "GM_QR_Q"
    R = "GM_QR_R"
    temp_table = "GM_QR_TEMP"
    I = "GM_QR_DECOMPOSE_IDENTITY"
    print 'Performing QR Algorithm. Max Iters=%s, Stop threshold=%s' % (steps

    cur = db_conn.cursor();

    gm_sql_table_drop_create(db_conn, EVal,"row_id integer, col_id integer, v
    gm_sql_table_drop_create(db_conn, EVec,"row_id integer, col_id integer, v

    gm_sql_table_drop_create(db_conn, I,"row_id integer, col_id integer, valu
    gm_sql_load_table(db_conn, I, [str(i) + " " + str(i) + " " + str(1) for i

    cur.execute("INSERT INTO %s" % (EVal) + " SELECT * FROM %s" % (T))
    db_conn.commit()

    for i in range(1,steps+1):

        try:
            gm_eigen_QR_decompose(EVal, n, Q, R)
        except psycopg2.DataError:
            db_conn.commit()
            break

```



```

gm_sql_table_drop_create(db_conn, EVal,"row_id integer, col_id integer, value float")

# Set EVal as RQ
gm_sql_mat_mat_multiply (db_conn, R, Q, EVal, "col_id", "row_id", "value", "row_id", "col_id", "value")

if i==1:
    # Copy Q to EVec
    cur.execute("INSERT INTO %s" % (EVec) + " SELECT * FROM %s" % (Q))
    db_conn.commit()
else:
    # Set EVec = EVec * Q
    gm_sql_table_drop_create(db_conn, temp_table,"row_id integer, col_id integer, value float")
    gm_sql_mat_mat_multiply (db_conn, EVec, Q, temp_table, "col_id", "row_id", "value", "row_id", "col_id", "value")

    gm_sql_table_drop_create(db_conn, EVec,"row_id integer, col_id integer, value float")
    cur.execute("INSERT INTO %s" % (EVec) + " SELECT * FROM %s" % (temp_table))
    db_conn.commit()

    cur.execute("SELECT max(abs(value)) FROM %s" % (EVec) + " WHERE row_id = %d" % (i))
    cur_err = cur.fetchone()[0]

    print "QR Algorithm Error = %s" % cur_err
    if cur_err <= err:
        break

cur.close()

# Drop temp tables
gm_sql_table_drop(db_conn, Q)
gm_sql_table_drop(db_conn, R)
gm_sql_table_drop(db_conn, temp_table)
gm_sql_table_drop(db_conn, I)

def gm_eigen (steps, num_nodes, err1, err2, adj_table=GM_TABLE_UNDIRECT):
    QR_max_iter = gm_param_qr_max_iter
    QR_stop_threshold = gm_param_qr_thres

```

```

basis_vect_0 = "GM_EG_BASIS_VECT0"
basis_vect_1 = "GM_EG_BASIS_VECT1"
next_basis_vect = "GM_EG_BASIS_VECT_NEXT"
temp_vect = "GM_EG_TEMP_VECT"
temp_vect2 = "GM_EG_TEMP_VECT2"
temp_vect3 = "GM_EG_TEMP_VECT3"
basis = "GM_EG_BASIS"
tridiag_table = "GM_EG_TRIDIAGONAL"
diag_table = "GM_EG_DIAG"
eigvec_table = "GM_EG_VEC"

cur = db_conn.cursor();
print "Computing Eigenvalues..."

# create basis vectors
gm_sql_vector_random(db_conn, basis_vect_1)
gm_sql_create_and_insert(db_conn, basis_vect_0, GMNODES, \
                        "id integer, value double precision", \
                        "id, value", "node_id, 0")

# Create table to store the basis vectors
gm_sql_table_drop_create(db_conn, basis, "row_id integer, col_id integer,

gm_sql_table_drop_create(db_conn, tridiag_table, "row_id integer, col_id i

beta_0 = 0
beta_1 = 0
alph_1 = 0

for i in range(1, steps+1):
    print "Iteration No: " + str(i)

    # Get the next basis
    gm_sql_table_drop_create(db_conn, next_basis_vect, "id integer, value
    gm_sql_adj_vect_multiply(db_conn, adj_table, basis_vect_1, next_basis
                        "id", "id", "value", "value", "src_id")

    alph_1 = gm_sql_vect_dotproduct (db_conn, next_basis_vect, basis_vect

    gm_sql_table_drop_create(db_conn, temp_vect, "id integer, value double
    # Orthogonalize with previous two basis vectors

```

```

cur.execute("INSERT INTO %s " % (temp_vect) +
            " (SELECT \"VECT_NEW\".id, " +
            " (\"VECT_NEW\".value - (%s * \"VECT_0\".value) - (%s
                                                    (beta_0, alph
            " FROM %s \"VECT_NEW\", %s \"VECT_0\", %s \"VECT_1\"
                                                    (next_basis_v
            " WHERE \"VECT_NEW\".id = \"VECT_0\".id AND \"VECT_0\"

db_conn.commit()

# Insert values into the tridiagonal table
cur.execute("INSERT INTO %s" % (tridiag_table) + " VALUES(%s,%s,%s)"
if i>1:
    cur.execute("INSERT INTO %s" % (tridiag_table) + " VALUES(%s,%s,%s)"
    cur.execute("INSERT INTO %s" % (tridiag_table) + " VALUES(%s,%s,%s)"

db_conn.commit()

# Save the basis vector
cur.execute("INSERT INTO %s " % (basis) +
            "SELECT id \"row_id\", %s \"col_id\", value " % (i) +
            "FROM %s" % (basis_vect_1))

db_conn.commit()

if i>1:
    gm_eigen_QR_iterate(tridiag_table, i, diag_table, eigvec_table, C

    for j in range(1,i+1):
        cur.execute("SELECT abs(value) FROM %s" % (eigvec_table) +
                    " WHERE col_id=%s AND row_id=%s" % (j,i))

        thr = cur.fetchone()
        if thr:
            thr = thr[0]
        else:
            thr = 0

        if thr <= err1:
            print "Performing SO with EigenVector " + str(j)
            # Get corresponding eigenvector
            gm_sql_table_drop_create(db_conn, temp_vect2,"id integer,

```

```

gm_sql_mat_colvec_multiply (db_conn, basis, eigvec_table,
                             "id", "value", "value", "value", "row_id")

# Selectively orthogonalize
r = gm_sql_vect_dotproduct (db_conn, temp_vect2, temp_vect3)

gm_sql_table_drop_create(db_conn, temp_vect3,"id integer,
cur.execute("INSERT INTO %s " % (temp_vect3) +
            " (SELECT \"VECT1\".id, " +
            " (\"VECT1\".value - (%s * \"VECT2\".value))
            " FROM %s \"VECT1\", %s \"VECT2\" " % (temp_vect2,
            " WHERE \"VECT1\".id = \"VECT2\".id)")

db_conn.commit()

gm_sql_table_drop_create(db_conn, temp_vect,"id integer,
cur.execute("INSERT INTO %s" % (temp_vect) + " SELECT * FROM %s" % (temp_vect2))

db_conn.commit()

beta_1 = gm_sql_normalize_vector (db_conn, temp_vect, "value");

if abs(beta_1) <= err2:
    break

# Prepare for next iteration
gm_sql_table_drop_create(db_conn, basis_vect_0,"id integer, value double
cur.execute("INSERT INTO %s" % (basis_vect_0) + " SELECT * FROM %s" % (temp_vect))
db_conn.commit()

gm_sql_table_drop_create(db_conn, basis_vect_1,"id integer, value double
cur.execute("INSERT INTO %s" % (basis_vect_1) + " SELECT * FROM %s" % (temp_vect))
db_conn.commit()

beta_0 = beta_1

# Get the eigen values and eigen vectors
gm_eigen_QR_iterate(tridiag_table, i, diag_table, eigvec_table, QR_max_iter)

gm_sql_table_drop_create(db_conn, GM.EIG.VALUES,"id integer, value double

```

```

print "Getting EigenValues..."
# Get top eigen values
cur.execute("INSERT INTO %s" % (GM_EIG_VALUES) +
            " SELECT col_id \"id\", value \"value\" FROM %s" % (diag_
            " WHERE col_id = row_id ORDER BY abs(value) desc")

db_conn.commit()

# Get the top k eigenvectors
print "Getting top %s EigenVectors..." % gm_param_eig_k
cur2 = db_conn.cursor();
gm_sql_table_drop_create(db_conn, GM_EIG_VECTORS,"row_id integer, col_id

cur.execute("SELECT id FROM %s ORDER BY abs(value) desc LIMIT %s " % (GM_
for idx in cur:
    i = idx[0]
    print "Getting eigenvector %s" % i
    gm_sql_table_drop_create(db_conn, temp_vect2,"id integer, value doubl
    gm_sql_mat_colvec_multiply (db_conn, basis, eigvec_table, temp_vect2,
                                "id", "value", "value", "value", "row_id", "c

    cur2.execute("INSERT INTO %s SELECT id \"row_id\", %s \"col_id\", val
                    " FROM %s" % (temp_vect2))

    db_conn.commit()

cur2.close()

#
# gm_sql_mat_mat_multiply (db_conn, basis, eigvec_table, GM_EIG_VECTORS, "
# "value", "row_id", "col_id", "r

print "EigenValues computed: "
gm_sql_print_table(db_conn, GM_EIG_VALUES)
#gm_sql_print_table(db_conn, GM_EIG_VECTORS)

cur.close()
# Drop temp tables
gm_sql_table_drop(db_conn, basis_vect_0)
gm_sql_table_drop(db_conn, basis_vect_1)
gm_sql_table_drop(db_conn, next_basis_vect)
gm_sql_table_drop(db_conn, temp_vect)

```

```

gm_sql_table_drop(db_conn, temp_vect2)
gm_sql_table_drop(db_conn, temp_vect3)
gm_sql_table_drop(db_conn, basis)
gm_sql_table_drop(db_conn, tridiag_table)
gm_sql_table_drop(db_conn, diag_table)
gm_sql_table_drop(db_conn, eigvec_table)

# Task 6: Fast Belief Propagation
# -----
def gm_belief_propagation(belief_file, delim, undirected, \
                          max_iterations = gm_param_bp_max_iter, stop_threshold = gm_param_bp_stop_threshold):

    next_table = "GM_BP_NEXT"
    print "Computing belief propagation..."

    # BP require that the graph is undirected.
    if (undirected):
        degree_col = "out_degree"
    else:
        degree_col = "out_degree+in_degree"

    cur = db_conn.cursor()
    cur.execute ("SELECT MAX(%s), SUM(%s), SUM((%s)*(%s))" % (degree_col, degree_col, degree_col),
                  "FROM %s" % GM_NODE_DEGREES)
    max_deg, sum_deg, sum_deg2 = cur.fetchone()

    c1 = 2+sum_deg
    c2 = sum_deg2 -1

    h = max(1 / (float)(2 + 2 * max_deg), sqrt((-c1 + sqrt(c1*c1 + 4*c2))/(8*sum_deg))
    print "Homophily factor = " + str(h)

    a = (4*h*h)/(1-4*h*h)
    c = (2*h)/(1-4*h*h)

    print "Getting the priors..."
    # Get the belief priors.
    gm_sql_table_drop_create(db_conn, GM_BELIEF_PRIOR, "node_id integer, belief float")
    gm_sql_load_table_from_file(db_conn, GM_BELIEF_PRIOR, "node_id, belief", belief_file)

    # Initialize belief table as belief priors

```

```

gm_sql_create_and_insert(db_conn, GM_BELIEF, GM_BELIEF_PRIOR, \
                        "node_id integer, belief double precision", \
                        "node_id, belief", "node_id, belief")

num_iterations = 0
while True:
    # Create Table to store the next belief
    gm_sql_table_drop_create(db_conn, next_table, "node_id integer, belief

    # Compute next belief
    cur.execute ("INSERT INTO %s " % next_table +
                " SELECT node_id, SUM(belief) FROM (" +
                " SELECT src_id \"node_id\", %s * SUM(belief)\" +
                " FROM %s, %s\" % (GM_TABLE_UNDIRECT, GM_BELIEF)\" +
                " WHERE dst_id = node_id GROUP BY src_id\" +
                " UNION ALL\" +
                " SELECT \"D\".node_id \"node_id\", %s*(%s)*b\" +
                " FROM %s \"D\", %s \"B\" \" % (GM_NODE_DEGREE)\" +
                " WHERE \"D\".node_id = \"B\".node_id\" +
                " UNION ALL\" +
                " SELECT * FROM %s \" % GM_BELIEF_PRIOR +
                " ) \"TAB\" GROUP BY node_id\" )

    db_conn.commit()

    diff = gm_sql_vect_diff(db_conn, GM_BELIEF, next_table, "node_id", "n

    # Recreate Belief table and copy values
    gm_sql_create_and_insert(db_conn, GM_BELIEF, next_table, \
                            "node_id integer, belief double precision", \
                            "node_id, belief", "node_id, belief")

    num_iterations = num_iterations + 1
    print "Iteration = %d, Error = %f" % (num_iterations, diff)

    if (diff <= stop_threshold or num_iterations > max_iterations):
        break

# Drop temp tables
gm_sql_table_drop(db_conn, next_table)

```

```

cur.close()

# Task 7: Triangle counting
# -----
def gm_naive_triangle_count(adj_table=GM_TABLE_UNDIRECT):

    temp_table = "GM_TRIANG_TEMP"
    temp_table2 = "GM_TRIANG_TEMP2"
    temp_table3 = "GM_TRIANG_TEMP3"

    cur = db_conn.cursor()
    gm_sql_table_drop_create(db_conn, temp_table, "row_id integer, col_id integer")
    gm_sql_table_drop_create(db_conn, temp_table2, "row_id integer, col_id integer")
    gm_sql_table_drop_create(db_conn, temp_table3, "row_id integer, col_id integer")

    # Copy the adjacency matrix
    cur.execute("INSERT INTO %s" % (temp_table) + \
                " SELECT src_id \"row_id\", dst_id \"col_id\", 1 \"value\" FROM %s" % adj_table)

    db_conn.commit()

    # Compute A^2
    gm_sql_mat_mat_multiply (db_conn, temp_table, temp_table, temp_table2, "count",
                             "value", "row_id", "col_id", "row_id", "col_id")

    # Compute A^3
    gm_sql_mat_mat_multiply (db_conn, temp_table2, temp_table, temp_table3, "count",
                             "value", "row_id", "col_id", "row_id", "col_id")

    cnt = gm_sql_mat_trace(db_conn, temp_table3, "row_id", "col_id", "value")

    print "Number of Triangles(naive) = " + (str(cnt/6))

    cur.close()

    # Drop temp tables
    gm_sql_table_drop(db_conn, temp_table)
    gm_sql_table_drop(db_conn, temp_table2)
    gm_sql_table_drop(db_conn, temp_table3)

```



```

def gm_eigen_triangle_count():

    cur = db_conn.cursor()
    #gm_eigen(steps, num_nodes, err1, err2, adj_table)
    print "Computing the count of triangles..."

    cur.execute("SELECT sum(value^3) FROM %s" % (GM_EIG_VALUES))
    cnt = cur.fetchone()[0]

    print "Number of Triangles = " + (str(cnt/6))

    cur.close()

# Innovative Task : Anomaly Detection for undirected graphs
def gm_anomaly_detection():
    cur = db_conn.cursor()
    gm_sql_table_drop_create(db_conn, GMEGONET,"node_id integer, edge_cnt in

    print "Extracting Features from Egonets"

    start_time = time.time()
    cur.execute("INSERT INTO %s " % (GMEGONET) +
                " SELECT node_id, sum(edge_cnt) \"edge_cnt\", sum(wgt
                " (SELECT \"T2\".dst_id \"node_id\", count(*)/2 \"edge
                " FROM %s \"T1\", %s \"T2\", %s \"T3\" " % (GM_TABLE
                " WHERE \"T1\".src_id = \"T2\".src_id AND \"T1\".dst_
                " GROUP BY \"T2\".dst_id" +
                " UNION ALL" +
                " SELECT src_id \"node_id\", count(*) \"edge_cnt\", s
                " FROM %s GROUP BY src_id) \"TAB\" " % (GM_TABLE_UNDI
                " GROUP BY node_id");

    db_conn.commit();
    print "Time taken = " + str(time.time()-start_time)

def main():
    global db_conn
    global GM_TABLE

```

```

# Command Line processing
parser = argparse.ArgumentParser(description="Graph Miner Using SQL v1.0")
parser.add_argument('--file ', dest='input_file ', type=str, required=True,
                    help='Full path to the file to load from. For weighted
                    graphs, the file should have the format (<src_id>, <dst_id>, <weight>).
                    If unweighted please run with --unweighted option.
                    delimiter other than "," (default), use --delim option.
                    NOTE: The file should have proper permissions set for
                    the postgres user.')
parser.add_argument('--delim ', dest='delimiter ', type=str, default=',',
                    help='Delimiter that separate the columns in the input file')

parser.add_argument('--unweighted ', dest='unweighted ', action='store_true',
                    help='For unweighted graphs. The input file should be in the format
                    (<src_id>, <dst_id>). For algorithms that require weights, a weight
                    of 1 will be used')

parser.add_argument('--undirected ', dest='undirected ', action='store_true',
                    help='Treat the graph as undirected instead of directed. If the
                    the input graph is first converted into an undirected graph with
                    with same weight. NOTE: Graph algorithms like eigen centrality,
                    connected components etc require undirected graphs and the
                    undirected version of the graph irrespective of whether the input
                    is directed or undirected')

parser.add_argument('--dest_dir ', dest='dest_dir ', type=str, required=True,
                    help='Full path to the directory where the output tables are
                    stored')

parser.add_argument('--belief_file ', dest='belief_file ', type=str, default='belief.prior',
                    help='Full path to belief priors file. The file should be in the format
                    (<node_id>, <belief>). Specify a different delimiter if needed.
                    The prior beliefs are expected to be centered around 0, positive
                    nodes >0, negative nodes <0 and unknown nodes 0')

args = parser.parse_args()

try:
    # Run the various graph algorithm below
    db_conn = gm_db_initialize()

    gm_sql_table_drop_create(db_conn, GM_TABLE, "src_id integer, dst_id integer, weight integer")
    if (args.unweighted):

```

```

        col_fmt = "src_id , dst_id"
    else:
        col_fmt = "src_id , dst_id , weight"

    gm_sql_load_table_from_file(db_conn, GM_TABLE, col_fmt, args.input_file)

    gm_to_undirected(False)

    if (args.undirected):
        GM_TABLE = GM_TABLE_UNDIRECT

    # Create table of node ids
    gm_create_node_table()
    # Get number of nodes
    cur = db_conn.cursor()
    cur.execute("SELECT count(*) from %s" % GM_NODES)
    num_nodes = cur.fetchone()[0]

    gm_node_degrees()

    # Tasks
    gm_degree_distribution(args.undirected)           # Degree distribution
    kcore(args)
    gm_pagerank(num_nodes)                             # Pagerank
    gm_connected_components(num_nodes)                 # Connected components
    gm_eigen(gm_param_eig_max_iter, num_nodes, gm_param_eig_thres1, gm_param_eig_thres2)
    gm_all_radius(num_nodes)
    if (args.belief_file):
        gm_belief_propagation(args.belief_file, args.delimiter, args.undirected)

    gm_eigen_triangle_count()
    #gm_naive_triangle_count()

    # Save tables to disk
    gm_save_tables(args.dest_dir, args.belief_file)
    #gm_anomaly_detection()

    gm_db_buylldb(db_conn)
except:
    print "Unexpected error:", sys.exc_info()[0]
    if (db_conn):

```

```
        gm_db_bubye(db_conn)

    raise

if __name__ == '__main__':
    main()
```

Contents

1	Introduction	1
2	Survey	2
2.1	Papers read by Ye Zhou	2
2.2	Papers read by Ye Zhou	2
2.3	Papers read by Ye Zhou	3
2.4	Papers read by Jin Hu	3
2.5	Papers read by Jin Hu	4
2.6	Papers read by Jin Hu	4
3	Proposed Method	5
4	Experiments	5
5	Conclusions	19
A	Appendix	20
A.1	Labor Division	20
A.2	Code for K-core	20