15-826 Project Phase 3 Report

Jiajun Wang jiajunwa@andrew.cmu.edu

San-Chuan Hung sanchuah@andrew.cmu.edu

November 25, 2014

1 Indexing Experiment

1.1 Indexing Experiment

In the experiment, we treated all graphs as direct ones. So "as-skitter.ungraph-75000.txt" is extended to a direct graph.
Tested graphs include: p2p-Gnutella31.txt, as-skitter.75000.txt, ca-AstroPh.txt, email-EuAll.txt, cit-HepTh.txt.

1.1.1 Degree distribution

Gm Node Degrees
The algorithm aggregates the count of dist nodes and src nodes in GM_TABLE for counting in degree and out degree.
We tried to add hash index and btree index on dist_id and src_id. The result shows that indices do not improve the performance. Actually the overhead of building the index makes the total running time longer.
One explanation of this result is that "group by" goes through all data. Index does not optimize the total sql running time in this case.

Index	p2p-Gnutella31.txt	as-skitter.75000.txt	ca-AstroPh.txt	email-EuAll.txt	cit-HepTh.txt
None	1.577036858	3.187309027	1.678581953	3.384658098	3.223124027
hash	2.431274891	4.133288145	4.991292953	10.31514502	3.931537151
btree	2.418382883	4.755445004	2.523052931	5.85737586	3.046495914

Gm Degree Distribution
The algorithm aggregates the count of out_degree and in_degree in GM_NODE_DEGREES for counting degree distribution.
We tried to build indices on out_degree and in_degree for testing whether these indices can accelerate the group by. Notice that the original count is very fast, so I change the code.

The group by sql will run 100 times for time estimating.
Basically we have 2 columns in the scope: in_degree and out_degree. We tried: 1. hash index on both columns; 2. btree index on both columns; 3. joint btree index on the columns; 4. joint btree index plus separate btree indices on both columns. The result shows that indices do not help the sql running.
The reason is the same as "gm_node_degrees". If the sql needs to go through the whole table anyway, indices do not improve the performance.

Index	p2p-Gnutella31.txt	as-skitter.75000.txt	ca-AstroPh.txt	email-EuAll.txt	cit-HepTh.txt
None	16.55248189	18.40389895	12.65532494	35.27958608	9.814982176
hash	14.32086205	20.65242195	9.01839304	50.09777999	9.146636009
btree	13.04618001	17.1775279	9.203239918	30.25537395	11.63468695
joint btree	12.15740585	17.67118001	11.81989694	31.76970196	9.897273064
all btree	12.51487803	12.83897305	12.14183187	31.41780305	8.73434186

1.1.2 PageRank

Index of columns in "WHERE" condition can help MySQL speed up value comparison in join operation. Therefore, we found that there are 5 possible positions to add index: GM_Table.src_id, norm_table.src_id, GM_PAGERANK.node_id, offset_stable.node_id, and next_table.node_id.
According to experiment result, we found that adding B-tree index on GM_PAGERANK.node_id improved the performance best. We also tried many index combinations, but the performance did not increase. Therefore, we decided to add B-tree index on index on GM_PAGERANK.node_id for Pagerank algorithm.

Index Type	**Index column**	**p2p-Gnutella31**	**ca-AstroPh**	**email-EuAll**	**cit-HepTh**	**as-skitter.75000**	**Improvement**
No Index		5.253519	4.519811	41.935326	12.899988	17.064826	(baseline)
Btree	GM_Table.src_id	3.482836	4.240341	28.786316	5.008184	12.756178	31.533%
Hash	GM_Table.src_id	5.645929	5.414358	33.585078	6.153968	14.20149	12.345%
Btree	norm_table.src_id	3.725514	7.023653	34.940364	5.243184	11.328331	16.667%
Hash	norm_table.src_id	4.831202	8.649363	37.06228	9.590399	11.593534	-2.797%
Btree	GM_PAGERANK.node_id	2.834585	4.510517	28.573334	5.31056	12.19424	33.097%
Hash	GM_PAGERANK.node_id	2.998954	6.758051	33.857276	5.976427	10.206583	21.303%
Btree	offset_stable.node_id	3.235787	5.560484	32.713935	9.890352	14.586771	15.044%
Hash	offset_stable.node_id	6.353038	6.031949	35.412756	4.819224	14.377134	7.912%
Btree	next_table.node_id	3.348404	7.015845	57.072034	4.326479	8.313848	12.537%
Hash	next_table.node_id	2.320558	4.440724	27.453469	4.527795	10.225985	39.417%

1.1.3 Weakly connected components

Firstly, the sql needs to update the component ids by comparing node ids based on link_table (GM_TABLE_UNDIRECT) in a loop. The component id is retrieved from the minimum node_id. So btree index should help.
Secondly, vector different is calculated based on node_id and component_id. So hash index on node_id column should help because there is a node_id = component_id condition in the sql.

Initially, we tried btree index on GM_CON_COMP.component_id. It does improve the performance. Then we add hash index on GM_CON_COMP.node_id. It turns out that the performance is improved again. After that, we tried to add hash index on the temp table and GM_TABLE_UNDIRECT table's columns. But the enhancement is not obvious.
So the 2 indices do work is btree on GM_CON_COMP.component_id and hash on GM_CON_COMP.node_id. For the first one, it mainly improves MAX() function. For the second one, it improves the "where" condition in sqls. After these 2 are added, other additional indices only increasing overhead instead of shorten the running time.

Index	**p2p-Gnutella31.txt**	**as-skitter.75000.txt**	**ca-AstroPh.txt**	**email-EuAll.txt**	**cit-HepTh.txt**
None	53.51399302	119.4098661	50.50897098	123.9283819	41.17333102
component_id(btree)	36.13925004	122.898526	39.09370708	149.132715	26.45658684
component_id(btree), node_id(hash)	25.45816708	86.28342104	22.88536716	125.9463222	27.85415602
component_id(btree), node_id(btree)	38.65054893	117.9664488	32.99039006	155.06496	31.62352514
component_id(btree), node_id(hash), temp.node_id(hash)	40.20039201	109.426384	28.04028392	133.7669752	30.60440493
component_id(hash), node_id(hash)	26.92220187	90.23280811	24.43618298	210.5891101	29.8577292
component_id(btree), node_id(hash), link_table_name.dst_id(hash)	28.92120504	83.67425203	26.90488887	138.1120729	30.00807405

1.1.4 Eigenvalue computation (via Lanczos-SO and QR algorithms)

Index of columns in "WHERE" condition can help MySQL speed up value comparison in join operation. Therefore, we found that there are 7 possible positions to add index: G.row_id + G.col_id, Q.row_id + Q.col_id, R.row_id + R.col_id, Eval.row_id + Eval.col_id, next_basis_vect.id, basis_vect_0.id, and basis_vect_1.id.
According to experiment result, we found that adding B-tree index on Eval.row_id and Eval.col_id improved the performance best. We also tried many index combinations, but the performance did not increase. Therefore, we decided to add B-tree index on index on Eval.row_id and Eval.col_id for Eigenvalue computation algorithm.

Index Type	**Index column**	**p2p-Gnutella31**	**ca-AstroPh**	**email-EuAll**	**cit-HepTh**	**as-skitter.75000**	**Improvement**
No cache		432.941635	42.247019	87.824524	34.531853	92.089166	(baseline)
Hash	G.row_id + G.col_id	183.792917	35.275701	76.680822	25.106503	83.686243	24.631%
Btree	G.row_id + G.col_id	212.340946	38.845005	95.304085	37.262694	137.772463	-1.405%
Hash	Q.row_id + Q.col_id	321.794434	28.335948	84.675914	37.241847	139.774482	0.511%
Btree	Q.row_id + Q.col_id	200.10483	28.930667	78.577032	34.081588	118.320216	13.729%
Hash	R.row_id + R.col_id	372.464409	41.754213	93.040661	34.649986	133.050895	-7.125%
Btree	R.row_id + R.col_id	282.987124	20.351836	56.000463	24.621752	72.129375	34.614%
Hash	Eval.row_id + Eval.col_id	373.817489	18.063141	60.053821	22.454597	80.1502	30.091%
Btree	Eval.row_id + Eval.col_id	66.592785	21.696901	58.072773	28.105395	99.974783	35.436%
Hash	next_basis_vect.id	683.642889	29.47404	87.384114	34.857184	81.874648	-3.404%
Btree	next_basis_vect.id	176.668195	22.145417	68.57327	38.863949	87.817504	24.157%
Hash	basis_vect_0.id	177.532078	41.937022	93.3785	39.510948	112.020925	3.468%
Btree	basis_vect_0.id	648.983299	20.541444	64.786556	25.261349	86.658346	12.090%
Hash	basis_vect_1.id	137.220548	21.072826	76.711587	26.999191	96.576688	29.603%
Btree	basis_vec_1.id	809.867548	26.501674	72.516903	27.561787	91.58051	-2.325%

1.1.5 Triangle count

This query is fully based on eigen value. And the aggregate function needs to go through all data. So index will not help.

Index	**p2p-Gnutella31.txt**	**as-skitter.75000.txt**	**ca-AstroPh.txt**	**email-EuAll.txt**	**cit-HepTh.txt**
None	0.000257969	0.000259876	0.000265121	0.00028801	0.000248194

1.1.6 K-core algorithm

Index of columns in "WHERE" condition can help MySQL speed up value comparison in join operation. Therefore, we found that there are 3 possible positions to add index: temp_degree_table.in_degree, temp_degree_table.node_id, and temp_link_table.src_id.
We tried all possible ways to add index on columns related to K-core algorithm; however, the experiment result showed that there was no significant improvement on executing time. Therefore, we decided not to add indices for K-core algorithm.

Index Type	Index column	p2p-Gnutella31	ca-AstroPh	email-EuAll	cit-HepTh	as-skitter.75000	Improvement
No cache		21.567541	35.087091	20.816424	80.608768	9.386785	(baseline)
Btreee	temp_degree_table.in_degree	30.652811	40.340461	23.426685	84.497394	6.604104	-8.963%
Hash	temp_degree_table.in_degree	28.243504	46.542064	32.58452	82.191118	13.88069	-33.994%
Btree	temp_degree_table.node_id	21.663131	37.245953	26.681098	103.65769	12.660117	-19.646%
Hash	temp_degree_table.node_id	21.885152	32.91817	25.688136	77.51908	6.447193	3.290%
Btree	temp_link_table.src_id	21.244998	46.501597	25.201246	86.955264	7.053383	-7.023%
Hash	temp_link_table.src_id	22.742343	40.598984	22.920802	90.440895	9.410922	-8.743%

1.1.7 Overall Validation

Next, we tested on several graphs about these indices. Some of them are sample graphs, some are new graphs for validating.
Most graph's processing time is shortened.
Notice that there is one graph's processing time becomes longer after we add indices. This might caused by the overhead for building the index on this graph.

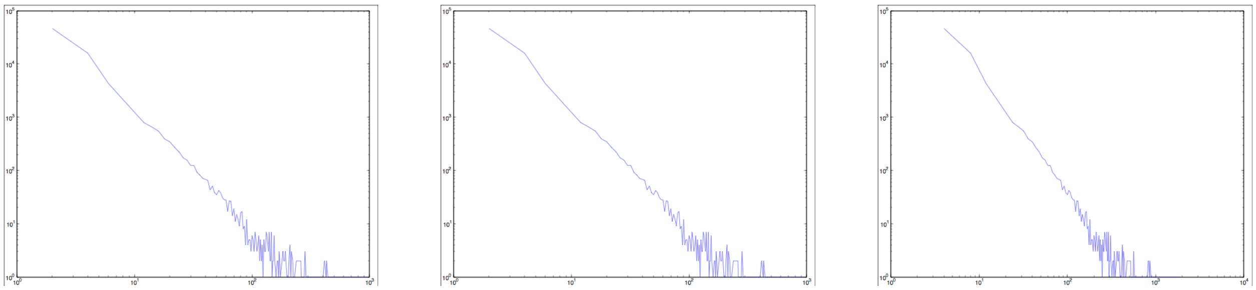
With indices or not	ca-AstroPh	cit-HepTh	email-EuAll	p2p-Gnutella31	soc-Slashdot0811
Run time with no indices	2m34.348s	3m6.760s	8m34.530s	6m58.804s	6m33.274s
Run time with indices	1m50.844s	5m7.181s	7m27.525s	5m22.789s	4m7.349s

2 20 graphs Result

2.1 Experiment on 20 graphs

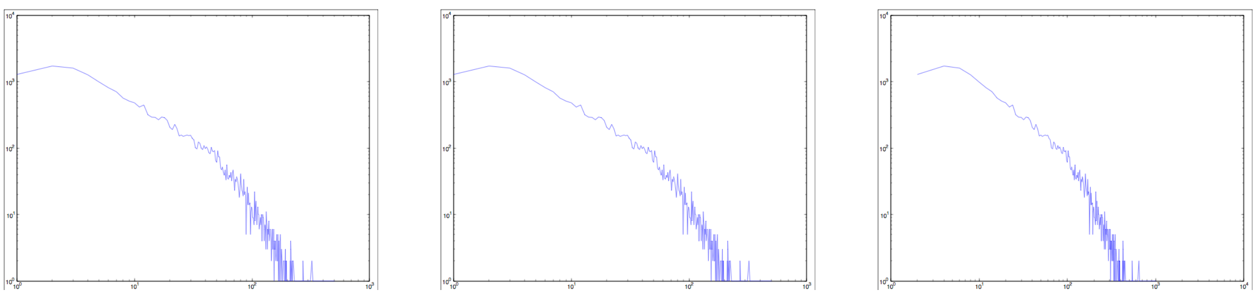
2.1.1 Degree distribution

Show in Figure 1 to 10.
The degree distributions in most of graphs follow power law. We found that there were spikes in p2p-Gnutella31 outdegree and degree distribution. The reason for the spike may be because of default peer number of setting in p2p software. Besides p2p graph, there were little spikes in email-Enron and soc-Slashdot0811.



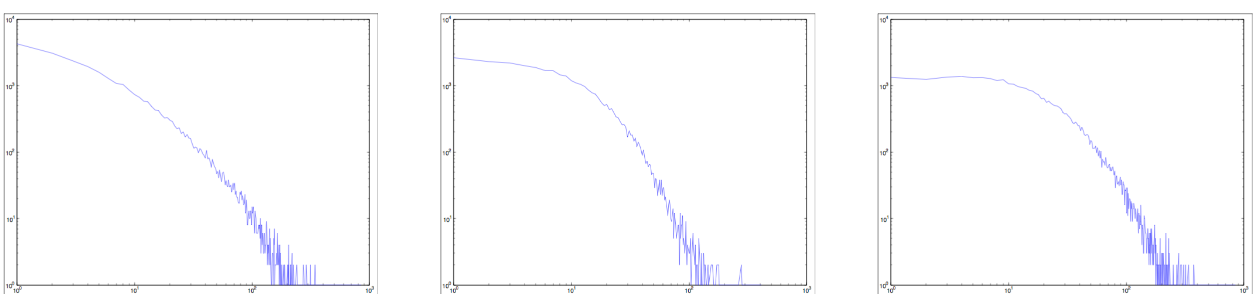
(a) In-Degree Distribution (b) Out-Degree Distribution (c) Degree Distribution

Figure 1: Degree Distributions of as-skitter



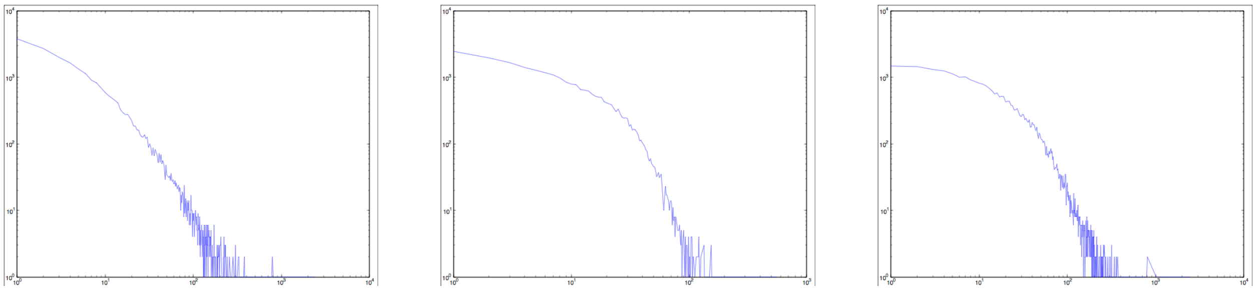
(a) In-Degree Distribution (b) Out-Degree Distribution (c) Degree Distribution

Figure 2: Degree Distributions of a-AstroPh



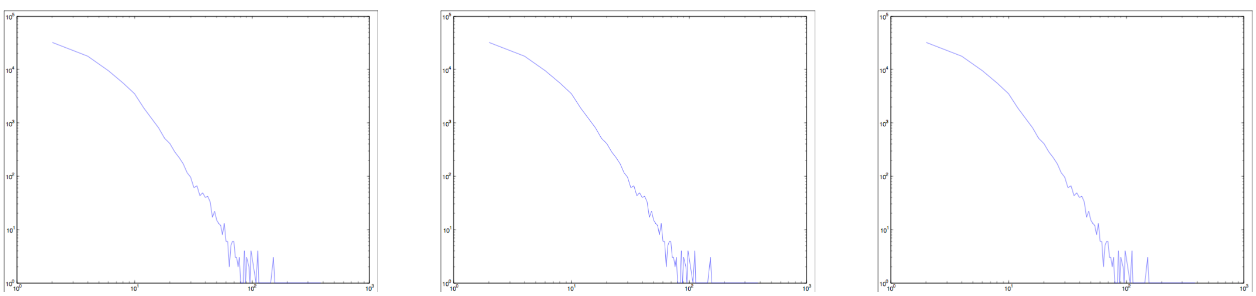
(a) In-Degree Distribution (b) Out-Degree Distribution (c) Degree Distribution

Figure 3: Degree Distributions of cit-HepPh



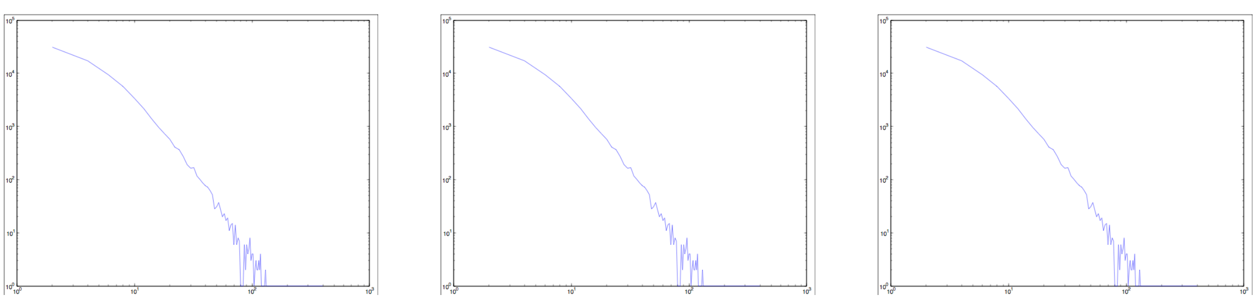
(a) In-Degree Distribution (b) Out-Degree Distribution (c) Degree Distribution

Figure 4: Degree Distributions of cit-HepTh



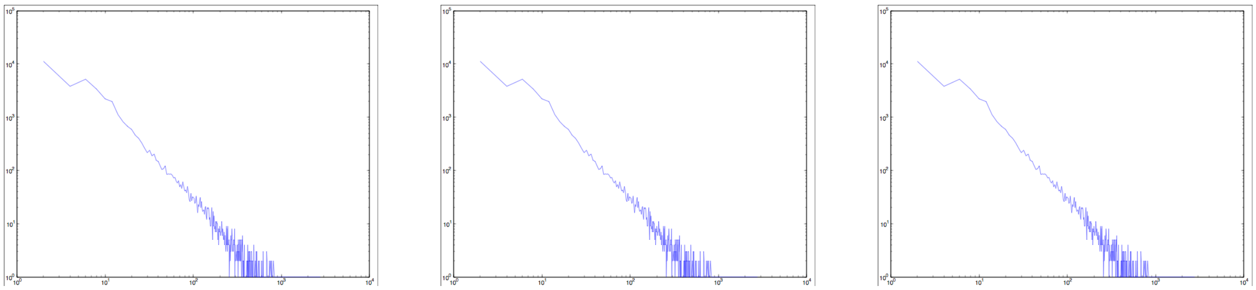
(a) In-Degree Distribution (b) Out-Degree Distribution (c) Degree Distribution

Figure 5: Degree Distributions of com-amazon.ungraph



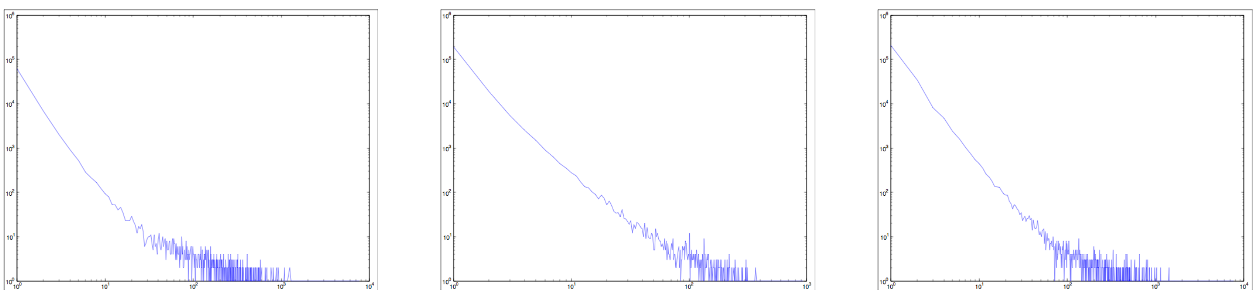
(a) In-Degree Distribution (b) Out-Degree Distribution (c) Degree Distribution

Figure 6: Degree Distributions of com-dblp.ungraph



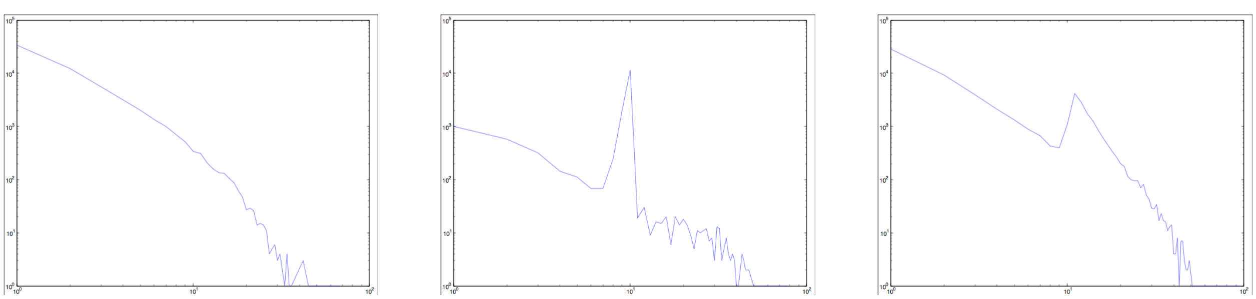
(a) In-Degree Distribution (b) Out-Degree Distribution (c) Degree Distribution

Figure 7: Degree Distributions of email-Enron.ungraph



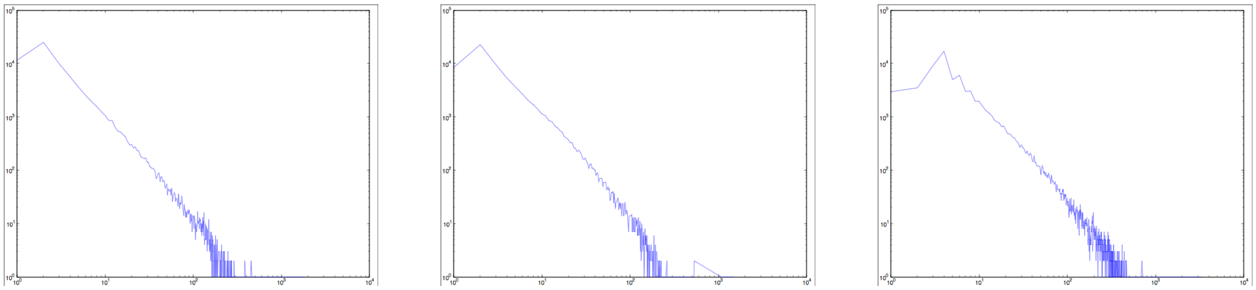
(a) In-Degree Distribution (b) Out-Degree Distribution (c) Degree Distribution

Figure 8: Degree Distributions of email-EuAll



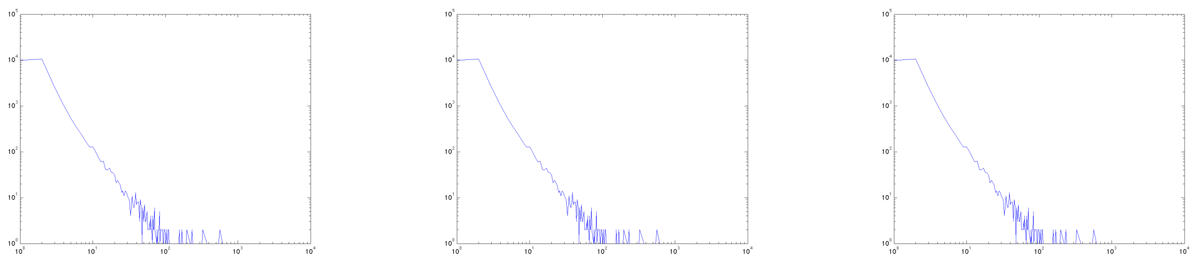
(a) In-Degree Distribution (b) Out-Degree Distribution (c) Degree Distribution

Figure 9: Degree Distributions of p2p-Gnutella31



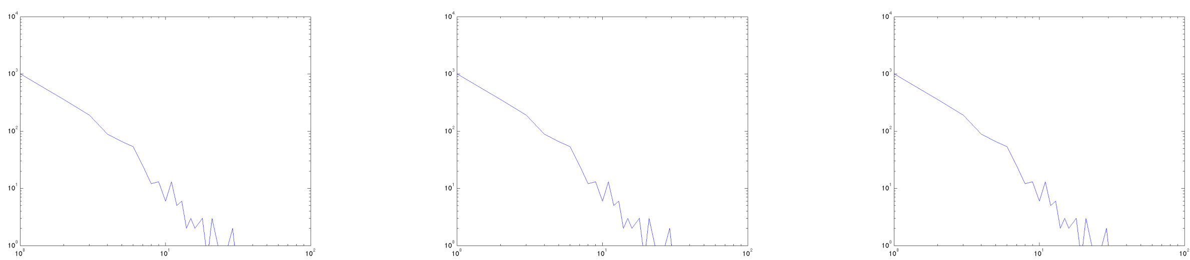
(a) In-Degree Distribution (b) Out-Degree Distribution (c) Degree Distribution

Figure 10: Degree Distributions of soc-Slashdot0811



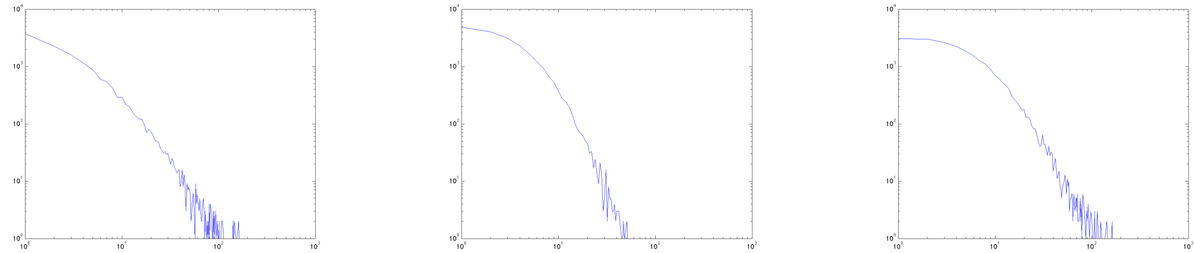
(a) In-Degree Distribution (b) Out-Degree Distribution (c) Degree Distribution

Figure 11: Degree Distributions of as-Caida



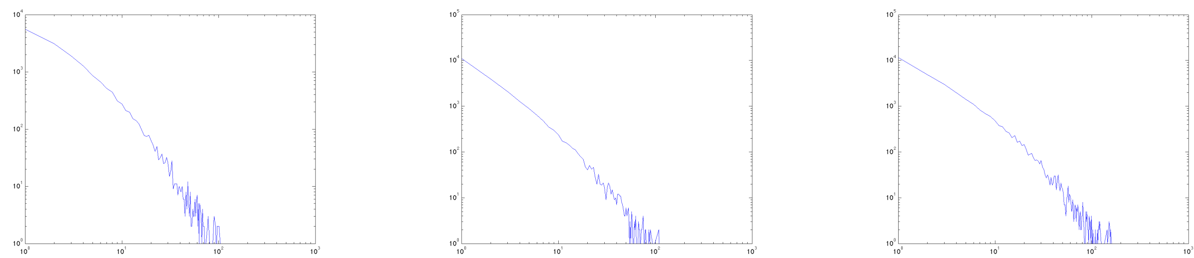
(a) In-Degree Distribution (b) Out-Degree Distribution (c) Degree Distribution

Figure 12: Degree Distributions of bio-protein



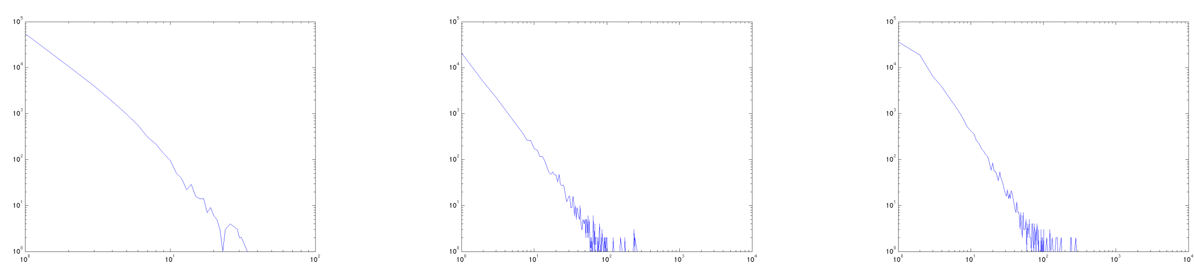
(a) In-Degree Distribution (b) Out-Degree Distribution (c) Degree Distribution

Figure 13: Degree Distributions of cit-Cora



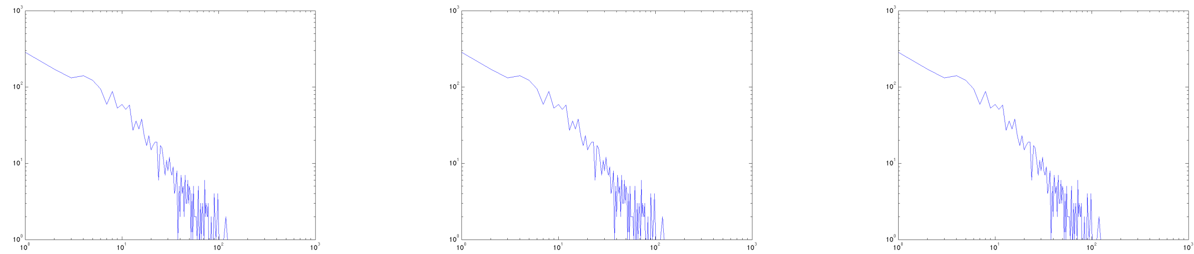
(a) In-Degree Distribution (b) Out-Degree Distribution (c) Degree Distribution

Figure 14: Degree Distributions of soc-digg



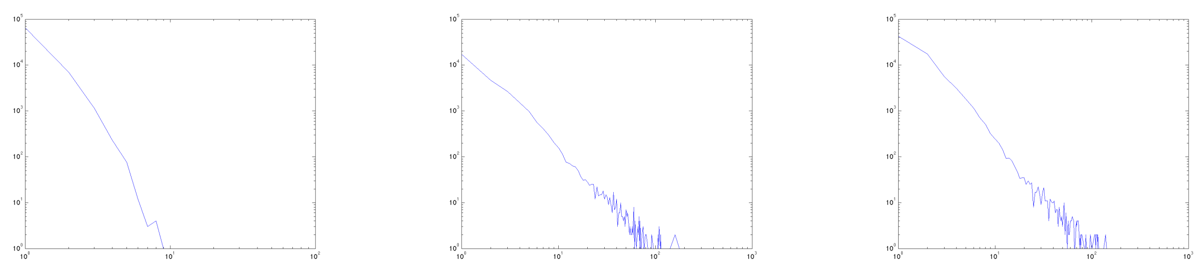
(a) In-Degree Distribution (b) Out-Degree Distribution (c) Degree Distribution

Figure 15: Degree Distributions of soc-flickr



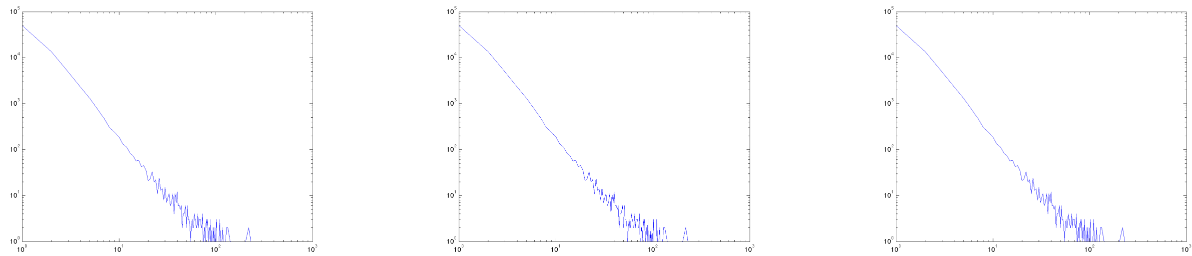
(a) In-Degree Distribution (b) Out-Degree Distribution (c) Degree Distribution

Figure 16: Degree Distributions of soc-hamsterster



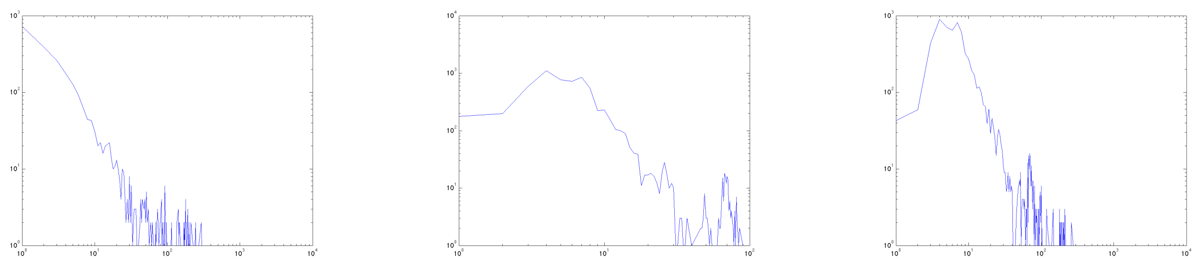
(a) In-Degree Distribution (b) Out-Degree Distribution (c) Degree Distribution

Figure 17: Degree Distributions of soc-pokec



(a) In-Degree Distribution (b) Out-Degree Distribution (c) Degree Distribution

Figure 18: Degree Distributions of soc-Youtube



(a) In-Degree Distribution (b) Out-Degree Distribution (c) Degree Distribution

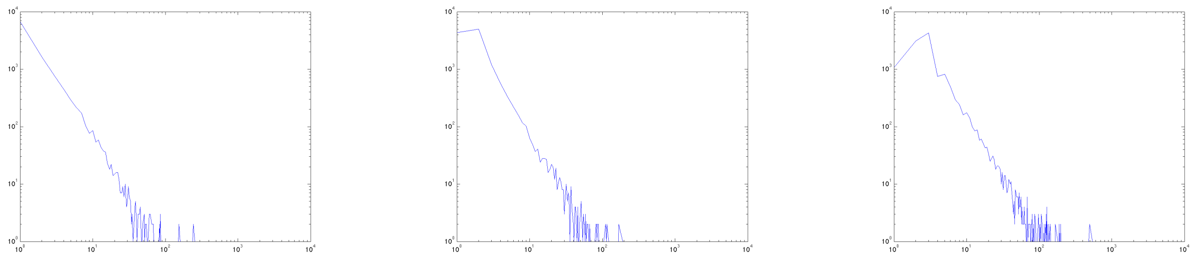
Figure 19: Degree Distributions of soft-jdkdependency

2.1.2 Weakly Connected Component and Triangle count

We find there are 2 highly connected graphs. p2p-Gnutella31 has 12 components and cit-HepPh has 61 components. This shows the nodes in these 2 graphs are mostly connected to each others. On the other hand, email-EuAll has 15836 components. Although the maximun component size is huge, the connection between nodes are not strong in this graph.
Triangle count shows that ca-AstroPh, cit-HepPh, email-Enron.ungraph are email-EuAll are densely connected. One interesting observation is that the p2p-Gnutella31 has very low triangle count, although the connected components shows that most nodes connect to each others in the graph. The may be caused by some very popular nodes. So that most nodes connect to these popular servers instead of connecting to each others.

Metrics	**as-skitter**	**ca-AstroPh**	**cit-HepPh**	**cit-HepTh**	**com-amazon**
components	310	290	61	143	1946
max group	69768	17926	34454	27465	47556
triangle	28389.34144	1061822.808	60696.51906	191035.2798	132.7590596

Metrics	**com-dblp**	**email-Enron**	**email-EuAll**	**p2p-Gnutella31**	**soc-Slashdot0811**
components	949	1065	15836	12	2091
max group	67361	33696	224832	62561	72780
triangle	786.338039	2059367.367	370075.0779	307.5803753	252186.8962



(a) In-Degree Distribution (b) Out-Degree Distribution (c) Degree Distribution

Figure 20: Degree Distributions of text-spanishbook

2.1.3 PageRank

The experiment result (Figure. 21) shows that the Pagerank value distribution in most of graphs follow power law.

2.1.4 Eigenvalue computation (via Lanczos-SO and QR algorithms)

	cit-HepPh	**com-amazon.ungraph-75000**	**com-dblp.ungraph-75000**	**email-Enron.ungraph**	**soc-Slashdot0811-75000**
First Eigenvalue	71.24869215	12.46375985	18.08047143	232.0718265	124.3376448
Second Eigenvalue	15.78100055	-10.49450063	-9.989018866	-52.74314639	-74.23852603
Third Eigenvalue	-11.28234916	2.528983265	3.950839681	16.07981363	3.277331459

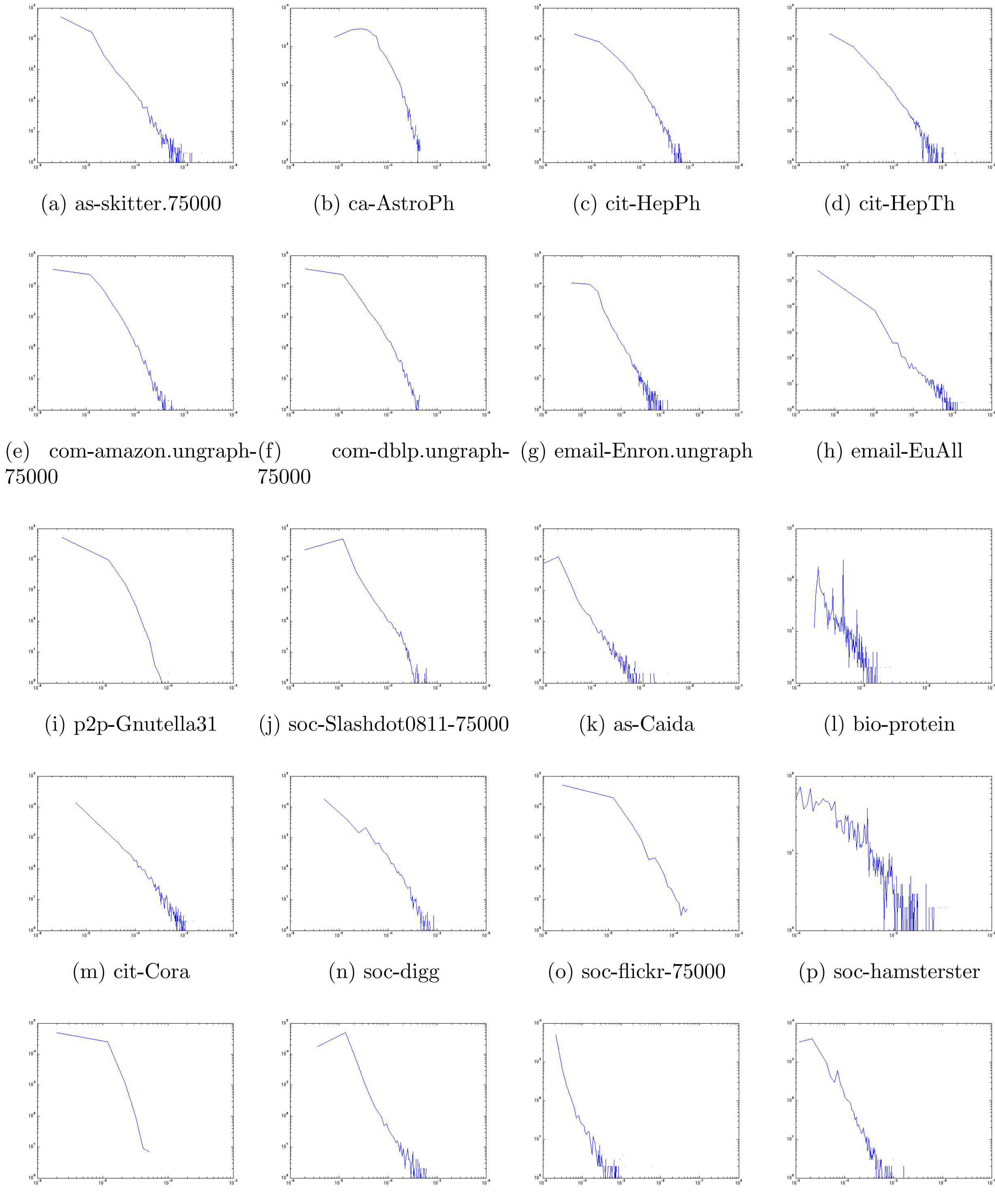
	p2p-Gnutella31	**ca-AstroPh**	**email-EuAll**	**cit-HepTh**	**as-skitter.75000**
First Eigenvalue	12.26069439	182.7509283	134.5445114	108.0148181	182.7509283
Second Eigenvalue	2.714800955	64.42806942	-59.94291686	-48.59746457	64.42806942
Third Eigenvalue	-2.60170164	-0.449525922	6.537952019	9.103275079	-0.449525922

	as-Caida.undir	**bio-protein-undir**	**cit-Cora**	**soc-digg**	**soc-flickr-75000**
First Eigenvalue	68.3693197062366	6.59050522092269	27.0826721488691	31.4948560594139	46.128302835195
Second Eigenvalue	-51.89780364579	-4.73938936859157	-9.39990749609768	3.5437520638839	-44.5991249417485
Third Eigenvalue	0.336124325308012	1.07545809389524	4.9442935842295	-3.43736182466055	1.53574108698456

	soc-hamsterster.undir	**soc-pokec-75000**	**soc-Youtube-75000.undir**	**soft-jdkdependency**	**text-spanishbook**
First Eigenvalue	45.2761211252026	9.38962714926706	15.7609882639711	143.265820052228	124.70922072648
Second Eigenvalue	-10.0662421612482	-9.28310856022425	-14.9067351906316	-126.79096036676	-92.5441274101652
Third Eigenvalue	5.6332490229777	0.918759752366816	1.16658315444281	2.26001659190722	8.30033058289115

2.1.5 K-core algorithm

We set k = 5 and apply K-core algorithm on 10 graphs. The result shows that the graphs can be grouped into two types of graphs. Type I graphs, like soc-Slashdot0811-75000, p2p-Gnutella31, email-EuAll, email-Enron.ungraph, cit-HepTh, cit-HepPh, ca-AstroPh, have a giant 5-core with more than 7000 nodes, which means that type I graph is densely connected. Type II graphs, like com-dblp.ungraph-75000, com-amazon.ungraph-75000, and as-skitter.75000, do not have such a giant k-core. Instead, there are many small (size $<$ 500) 5-cores in type II graphs. It indicates that Type II graphs are not densely connected.



(a) as-skitter.75000 (b) ca-AstroPh (c) cit-HepPh (d) cit-HepTh

(e) com-amazon.ungraph-75000 (f) com-dblp.ungraph-75000 (g) email-Enron.ungraph (h) email-EuAll

(i) p2p-Gnutella31 (j) soc-Slashdot0811-75000 (k) as-Caida (l) bio-protein

(m) cit-Cora (n) soc-digg (o) soc-flickr-75000 (p) soc-hamsterster

(q) soc-pokec-75000 (r) soc-Youtube-75000 (s) soft-jdkdependency (t) text-spanishbook

Figure 21: Pagerank value Distributions of 20 graphs

soc-Slashdot0811-75000		p2p-Gnutella31		email-EuAll		email-Enron.ungraph		cit-HepTh		cit-HepPh		ca-AstroPh	
size	frequency	size	frequency	size	frequency	size	frequency	size	frequency	size	frequency	size	frequency
26137	1	16174	1	7030	1	6	6	21181	1	28593	1	6	2
						7	2					7	2
						8	3					8	1
						9	1					11	1
						12	1					18	1
						15	1					12236	1
						11538	1						

as-Caida		cit-Cora		soc-digg		soc-flickr-75000		soc-hamsterster		soft-jdkdependency		ca-AstroPh	
size	frequency	size	frequency	size	frequency	size	frequency	size	frequency	size	frequency	size	frequency
1192	1	9355	1	6794	1	1122	1	1070	1	4869	1	3144	1

5-core distribution in Type I graphs

as-skitter.75000		com-dblp.ungraph-75000		com-amazon.ungraph-75000		bio-protein		soc-pokec		soc-Youtube-75000	
size	frequency	size	frequency	size	frequency	size	frequency	size	frequency	size	frequency
14	1	6	9	6	1	6	1	0	0	0	0
181	1	7	3								
		8	3								
		9	1								
		10	1								
		13	1								
		15	1								

5-core distribution in Type II graphs

3 Division of Labour

- Jiajung Wang:
  - Degree Distribution optimization and experiment
  - Weakly Connected Component optimization and experiment
  - Triangle count optimization and experiment
  - Organize latex files
- San-Chuan Hung:
  - PageRank optimization and experiment
  - K-core optimization and experiment
  - Eigenvalue computation optimization and experiment
  - Organize latex files