CPA Project

Erdogan Kevin, Katia Amichi

April 2019

1 Handling a large graph

Our setup for running times for this part and further is : i5-7440HQ CPU @ 2.8GHz.
Git link to our code : `https://github.com/zeeer0/cpa_project`

1.1 A special quantity

The following results were provided by the function "specialQuantity()", we have first cleaned the graph file and then calculate the quantity.

Special Quantity		
Graph Name	$\mathcal{Q}_G$	Running time(s)
Email-Eu-core	88 109 182	0.003146
Amazon	103 415 531	0.1827
Live Journal	789 000 450 609	4.820176
Orkut	22 292 678 512 329	15.769974
Friendster	379 856 554 324 947	474.847368

1.2 Degree distribution

For this part, we have generated a file named "degreeDistribution.txt" thanks to the function "degreeDistribution()" for each graph and then plot it with gnuplot to have the following results.

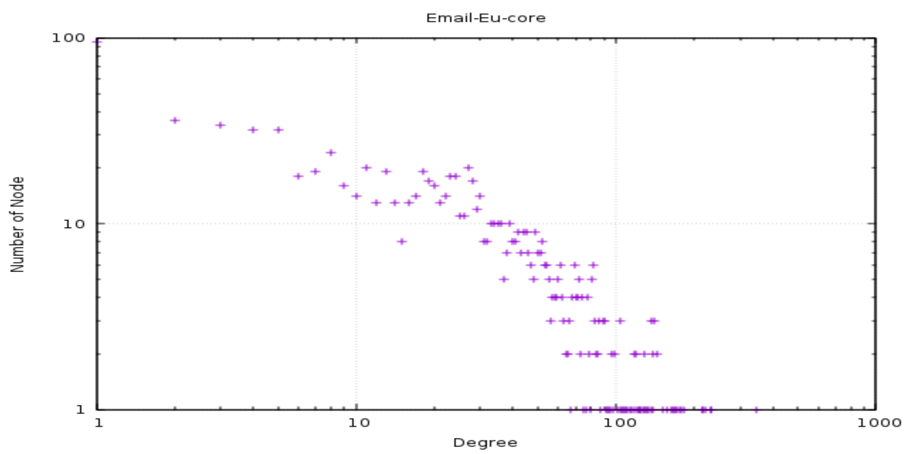


Figure 1: Degree distribution for Email-Eu-Core

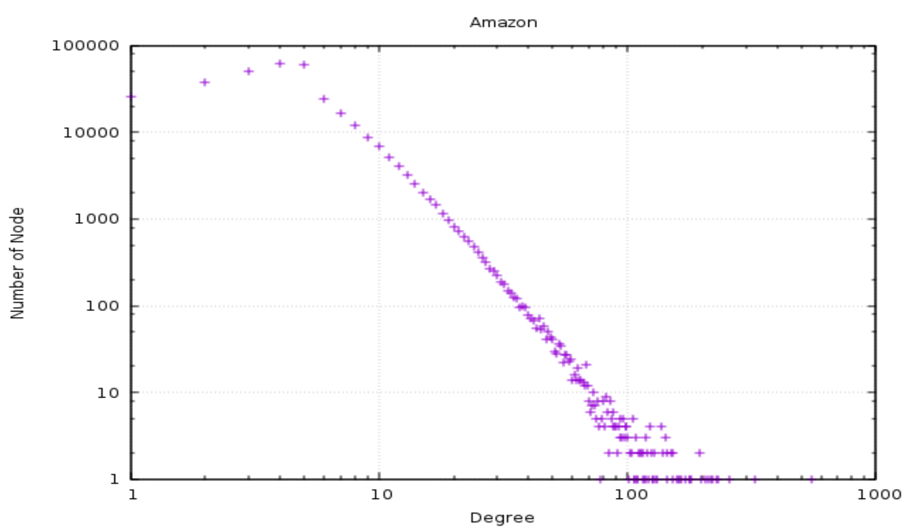


Figure 2: Degree distribution for Amazon

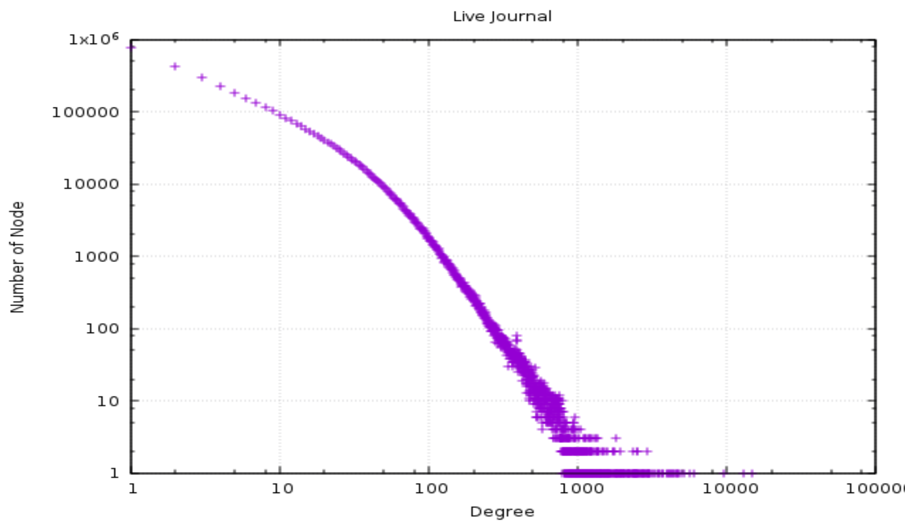


Figure 3: Degree distribution for Live Journal

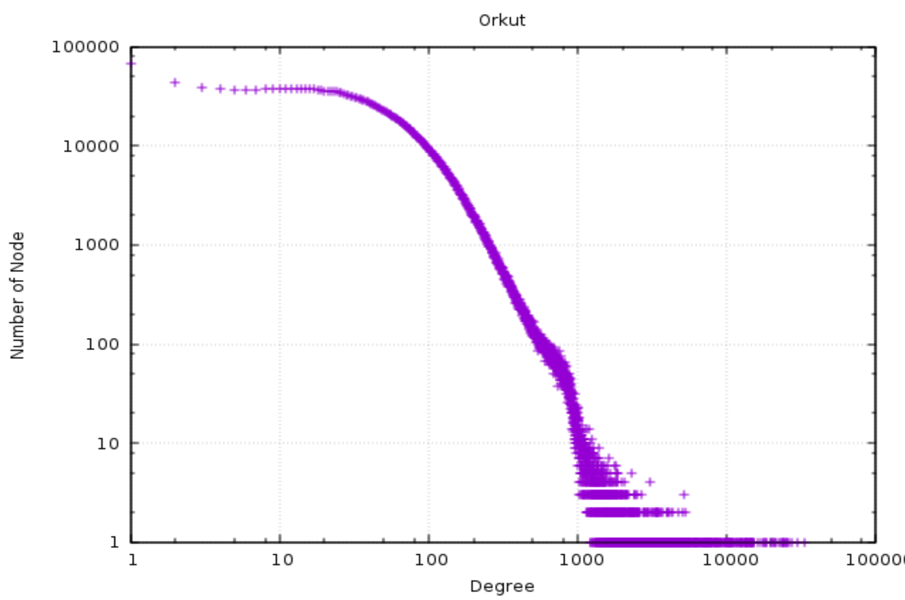


Figure 4: Degree distribution for Orkut

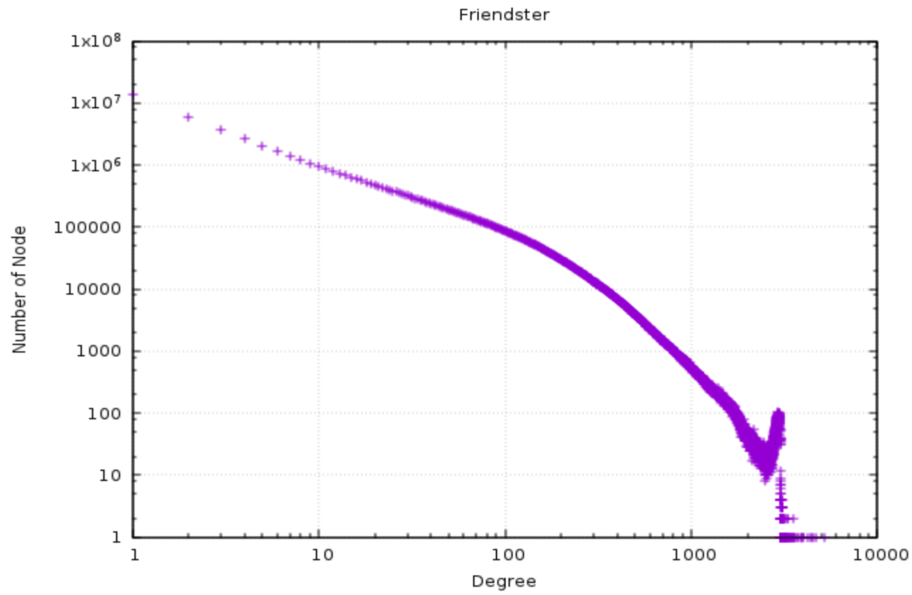


Figure 5: Degree distribution for Friendster

1.3 Three graph datastructures

Firstly, for the List of Edge datastructure, it is a simple way to store a graph in memory but very not efficient for the search of neighbours.
Amount of memory required : $sizeof(int)$ x 2 x $NumberOfEdges$ bytes.
For example, Friendster has 1 806 067 135 edges, so we need 14.5 Gig of RAM for storing the graph in memory.

Secondly, for the Adjacency Matrix datastructure, we have access to neighbours of a node very quickly but the required amount of memory for storing the graph fastly becomes creepy.
Amount of memory required : $sizeof(int)$ x $NumberOfNodes^2$ bytes.
In our example, we need up to 17 217 830 Gig of RAM...

And finally, for the Adjacency Array datastructure, it is the best way of the three datastructures for storing a huge graph because we need (approximately) the same amount of memory as the List of Edge datastructure but we have a very simple and efficient way to access to the neighbours of a node.
Amount of memory required : $sizeof(int)$ x 2 x $NumberOfEdges + C$ bytes, where C is a negligible amount of memory.

1.4 Breadth-First Search

For this exercise, we could not run the program on Friendster graph because it requires at least 16Gig of free RAM to store it in memory.
The results were provided by the function "max connections diameter()".

Listing Triangles		
Graph Name	Fraction of Nodes in the largest connected component	Lower Bound
Email-Eu-core	986	7
Amazon	334863	47
Live Journal	3997962	21
Orkut	3072441	9
Friendster	null	null

1.5 Listing triangles

For Friendster graph, same as above.
The results are given by the function "numberOfTriangle()", they are calculated by storing the graph in a Adjacency Array datastructure.

Listing Triangles		
Graph Name	Number of Triangles	Running time
Email-Eu-core	105 461	0.012474
Amazon	667 129	0.129404
Live Journal	177 820 130	36.073947
Orkut	627 584 181	392.643731
Friendster	null	null

2 Practical Work - Community Detection

2.1 Simple Benchmark

With p=0.6 and q=0.001

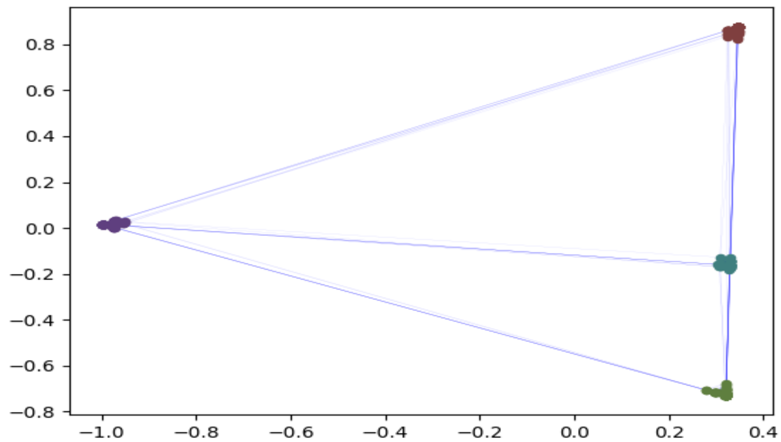


Figure 6: Graphs with various values of p and q

With p=0.6 and q=0.1

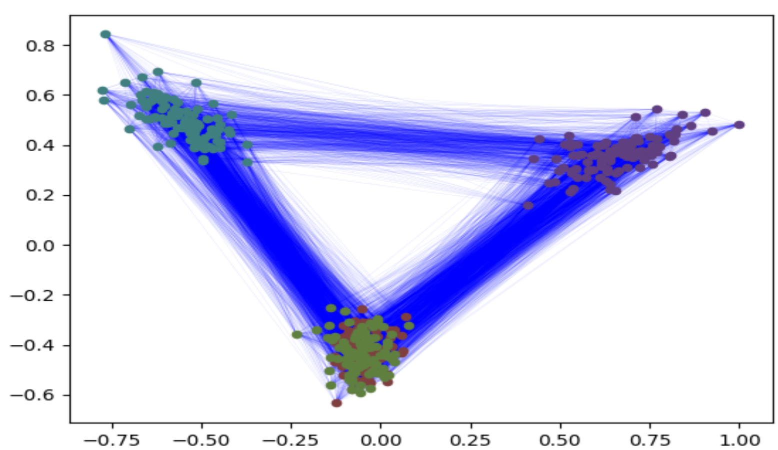


Figure 7: Graphs with various values of p and q

With p=0.6 and q=0.7

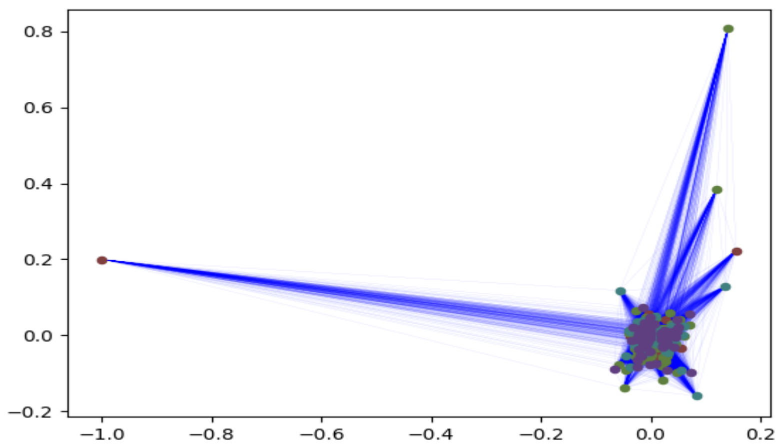


Figure 8: Graphs with various values of p and q

We notice that the more we diminish the value of q, the more we can distinguish the communities, as we can see on the different plots, when we increase the value of q we can see the separation between the communities.

2.2 Label propagation

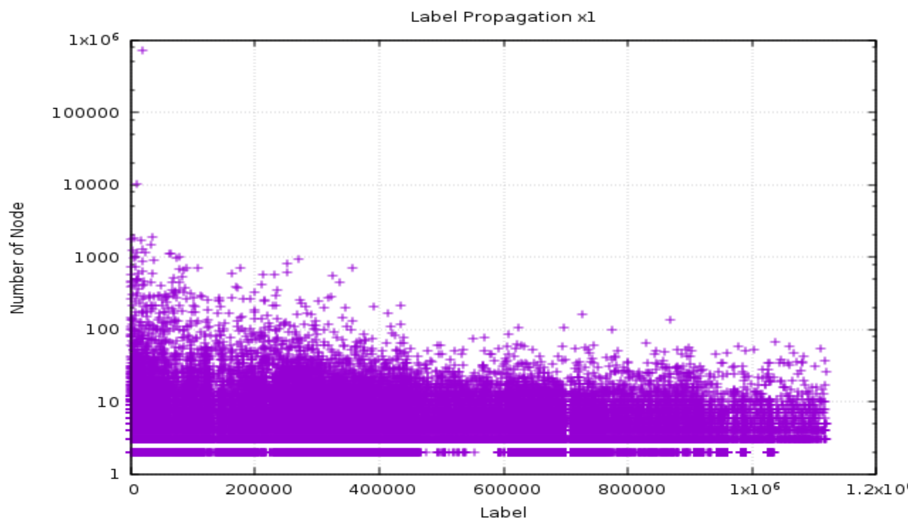


Figure 9: Histogram of the sizes of the communities

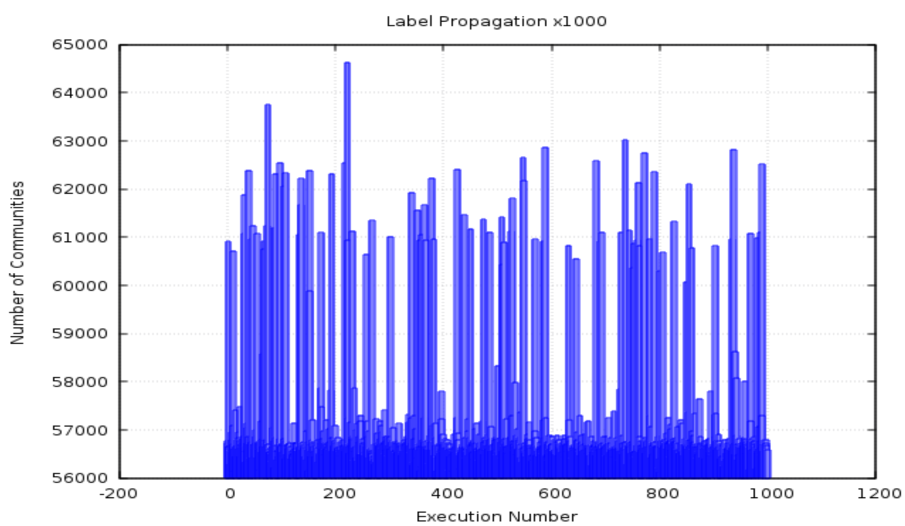


Figure 10: Histogram of the numbers of communities

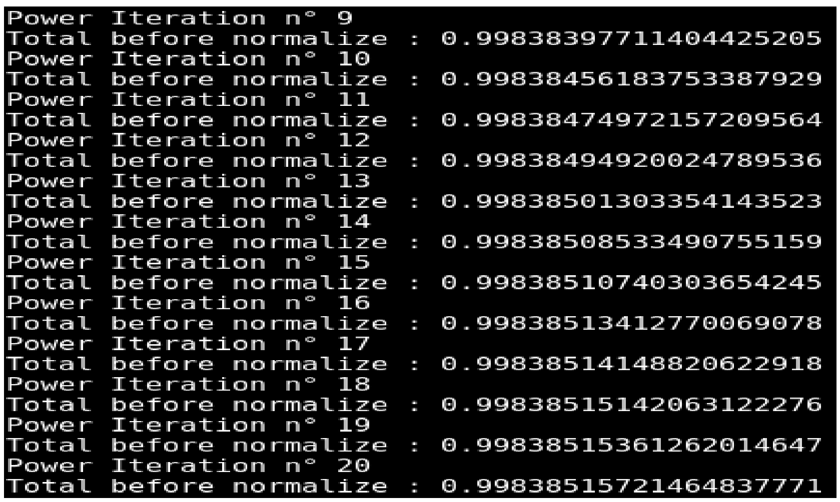
For the Figure 9, we can see a large amount of communities but they are, for the most, not exceeding 100 nodes.
More over in Figure 10, thanks to the random shuffling, we have each time

another amount of communities for each execution. But we can see that there is a range which is [55000:65000] that the number of communities stays in.

3 Practical Work - Pagerank

3.1 PageRank (directed graph)

3.1.1 Convergence



```
Power Iteration n° 9
Total before normalize : 0.99838397711404425205
Power Iteration n° 10
Total before normalize : 0.99838456183753387929
Power Iteration n° 11
Total before normalize : 0.99838474972157209564
Power Iteration n° 12
Total before normalize : 0.99838494920024789536
Power Iteration n° 13
Total before normalize : 0.99838501303354143523
Power Iteration n° 14
Total before normalize : 0.99838508533490755159
Power Iteration n° 15
Total before normalize : 0.99838510740303654245
Power Iteration n° 16
Total before normalize : 0.99838513412770069078
Power Iteration n° 17
Total before normalize : 0.99838514148820622918
Power Iteration n° 18
Total before normalize : 0.99838515142063122276
Power Iteration n° 19
Total before normalize : 0.99838515361262014647
Power Iteration n° 20
Total before normalize : 0.99838515721464837771
```

Figure 11: Number of iterations to reach convergence

As we see in Figure 11, we reached convergence after about 15 iterations with $\alpha = 0.15$.

3.1.2 Pagerank Results

For this part, we have implemented the algorithm of Power Iteration and run it 20 times to get the following results.

Highest and Lowest Pagerank		
Place	Highest Pagerank	LowerPageRank
1	United States	Aberdeen (disambiguation)
2	United Kingdom	Animal (disambiguation)
3	Germany	Antigua and Barbuda
4	2007	AWK (disambiguation)
5	2006	Demographics of American Samoa

3.2 Correlations

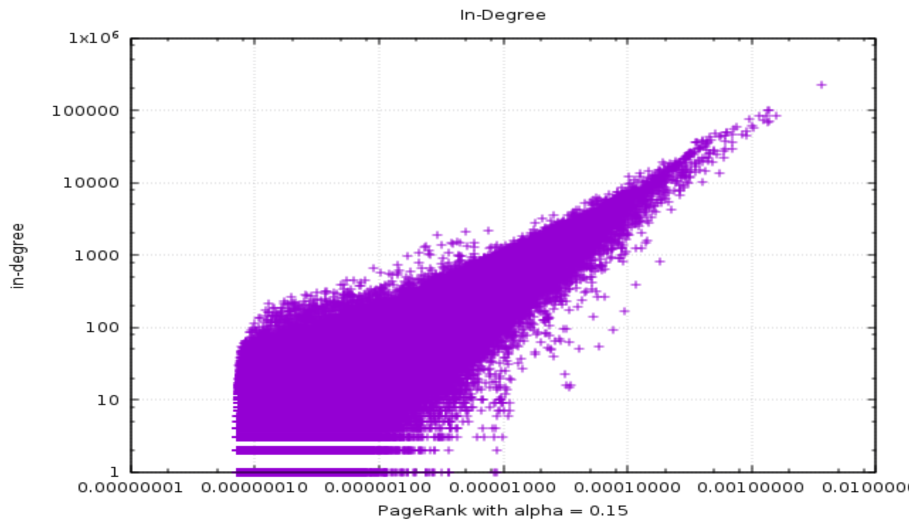


Figure 12: Scatter plot with y = in-degree

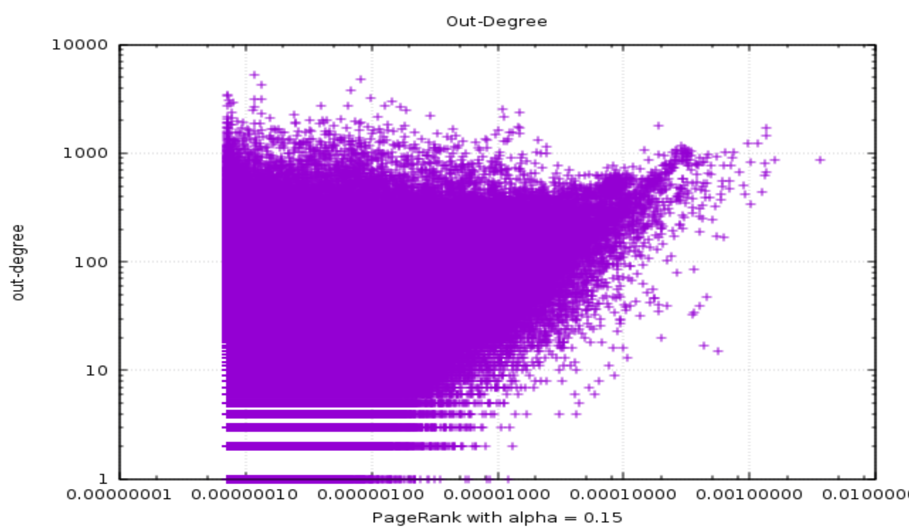


Figure 13: Scatter plot with y = out-degree

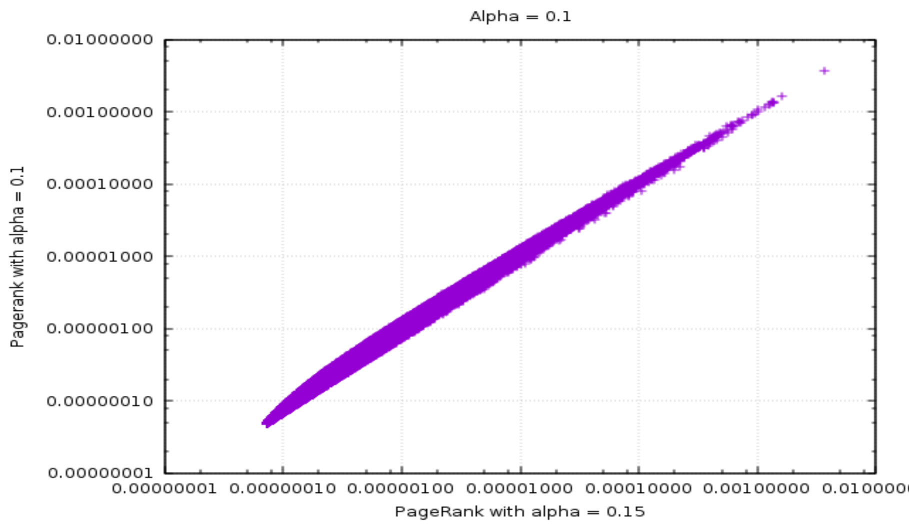


Figure 14: Scatter plot with y = Pagerank with $\alpha = 0.1$

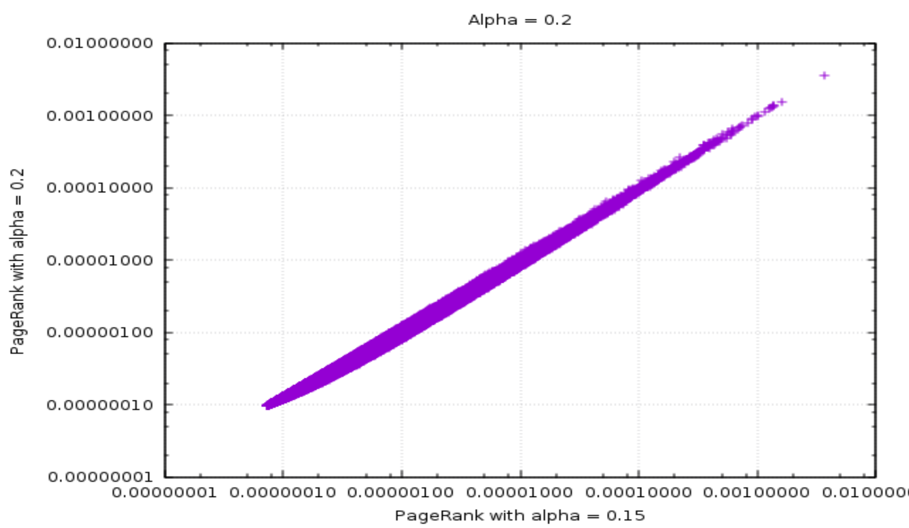


Figure 15: Scatter plot with y = Pagerank with $\alpha = 0.2$

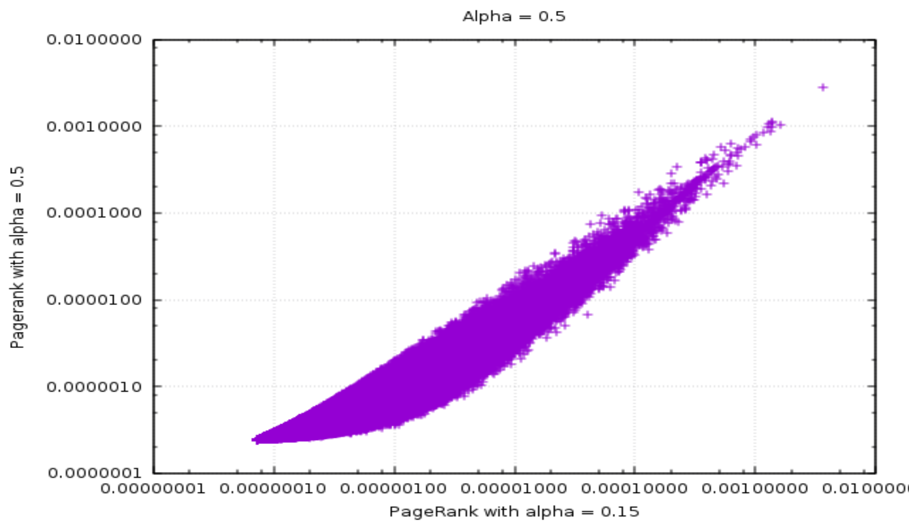


Figure 16: Scatter plot with y = Pagerank with $\alpha = 0.5$

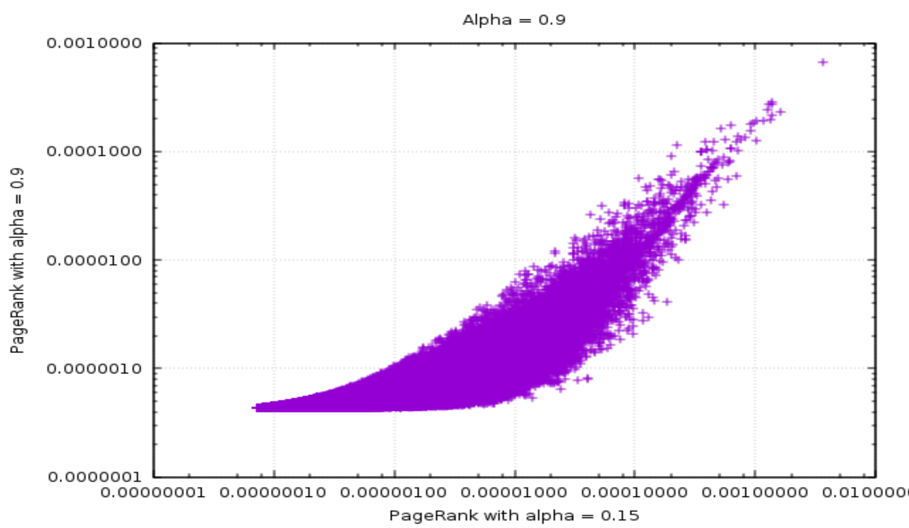


Figure 17: Scatter plot with y = Pagerank with $\alpha = 0.9$

3.2.1 Response

We used log scales because the Pageranks were too small.
About correlations, when α_1 and α_2 are close in x-axis and y-axis, it seems to be a linear function (Pageranks are very similar) but when α_2 in y-axis increase, the Pageranks in y-axis seems to be lower.

4 Densest subgraph

4.1 k-core decomposition

File	Core	The average degree density	The edge density	The size of a densest core ordering prefix
Email-Eu-core	34	27.352942	0.147059	187
Amazon	6	3.684211	0.204678	19
Live Journal	360	189.463547	0.494683	384
Orkut	259	227.834030	0.008583	26546
Friendster	null	null	null	null

4.2 Graph mining with k-core

We notice that there is a group of authors who have a core of 14 and who do not have a high degree and that all its author are of the same nationality.

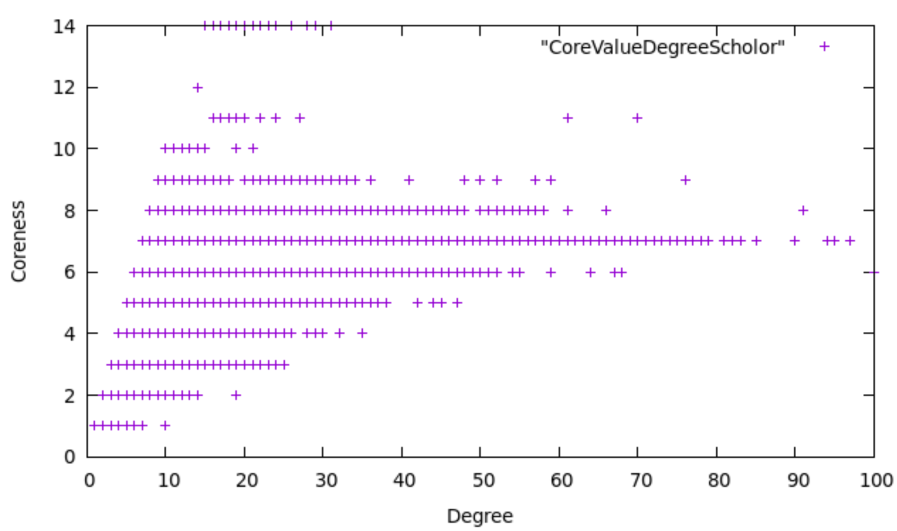


Figure 18: Core Value Degree for Scholor

With the logarithmic scales we notice that there is a limit that does not exceed.

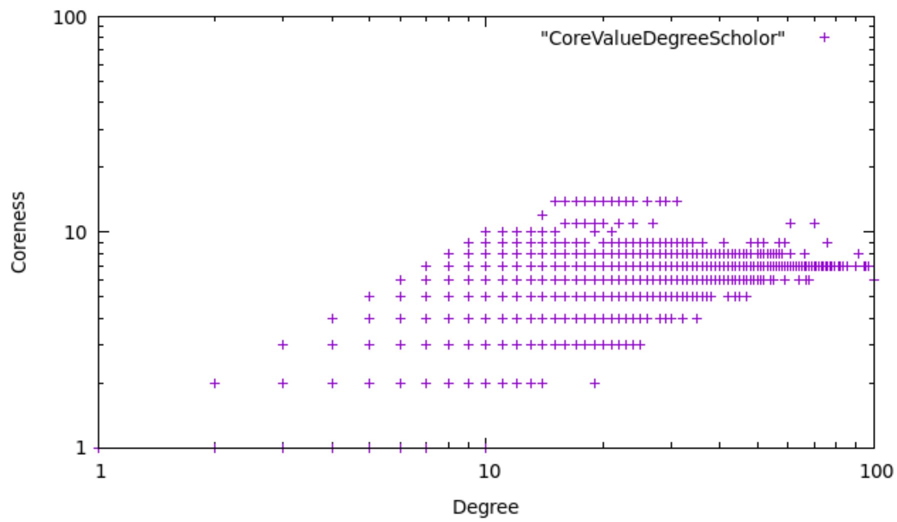


Figure 19: Core Value Degree for Scholor Log

We get the following results :

- k-core: 14
- Average degree density = 9.84719
- Edge density = 0.94215
- Size of a core ordering prefix = 28

The authors : They are authors who are quoted a lot between them. Sa-kwang, Sung-Pil, Chang-Hoo, Yun-soo, Hong-Woo, Jinhyung, Hanmin, Do-Heon, Myunggwon, Won-Kyung, Hwamook, Minho, Won-Goo, Jung, Dong-min, Mi-Nyeong, Sung, Minhee, Sungho, Seungwoo, Heekwan, Jinhee, Taehong, Mikyoung ,Ha-neul ,Seungkyun, Yun-ji.

4.3 Densest subgraph

We note that the more we increase the number of iteration, the closer we get to the values obtained on the first exercise.

4.3.1 t = 10

The number of iterations t to 10			
File	The average degree density	The edge density	The size of a densest core ordering prefix
Email-Eu-core	28.0629	2.30952	70
Amazon	4.8850	0.01888	244
Live Journal	199.8502	0.243066	232
Orkut	249.980	0.0587221	397

4.3.2 t = 100

The number of iterations t to 100			
File	The average degree density	The edge density	The size of a densest core ordering prefix
Email-Eu-core	27.0995	2.30952	185
Amazon	4.7760	0.0998711	107
Live Journal	191.082	0.243066	382
Orkut	228.827	0.0541537	1384

4.3.3 t = 1000

The number of iterations t to 1000			
File	The average degree density	The edge density	The size of a densest core ordering prefix
Email-Eu-core	27.4493	0.241783	229
Amazon	4.7760	0.0998711	98
Live Journal	4.7556	0.0984641	441
Orkut	nul	l null	null

For 1000 iterations, it does not end in reasonable time.

We consider the dense subgraph problem that extracts a subgraph with a prescribed number of vertices that has the maximum number of edges in a given graph.

Assuming that the density of the optimal output subgraph is high, where density is the ratio of number of edges to the number of edges in the clique on the same number of vertices, with proving that 2 * density score increases the max degree of the graph.

4.4 Graph not fitting in main memory

For this exercise, we implemented the same algorithm as Exercise 3 but each time running the file without storing it. Although we keep nothing in memory, the slowest part is the file reading (which we do several times).