Clemson University
CPSC 8480 - Network Science Homework I
Roshan Bhandari

Part 2:

Downloaded Gephi and visualized power grid data available in gephi samples, it looked like below:

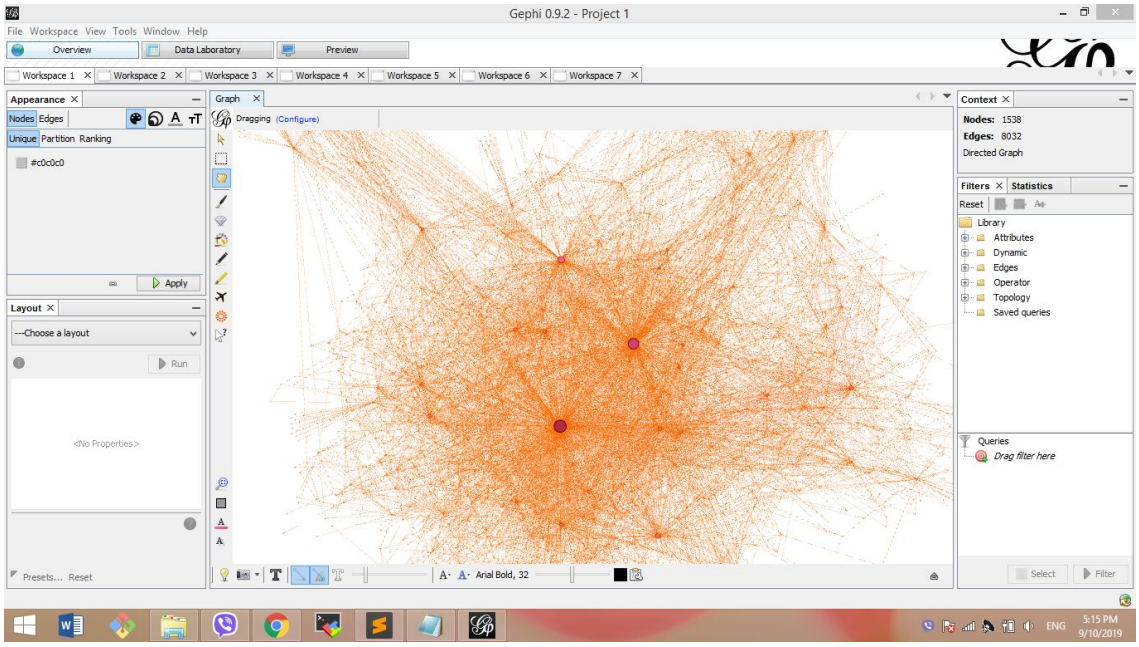

Fig: Gephi Visualization of power grid sample file

Part 3

DataSet: Amazon co-purchase Network from SNAP (Stanford)

In this paper, I try to analyze the amazon co-purchase dataset published by stanford university[1]. Amazon co-purchase dataset comprises of around 153439 nodes or products and 544771 edges. These edges represent co-purchase based on C*ustomers Who Bought This Item Also Bought* feature on Amazon website. The data was collected on March 02, 2003.[2]

1. **Co-Purchase Network (subgraph) visualization in python using NetworkX Library**

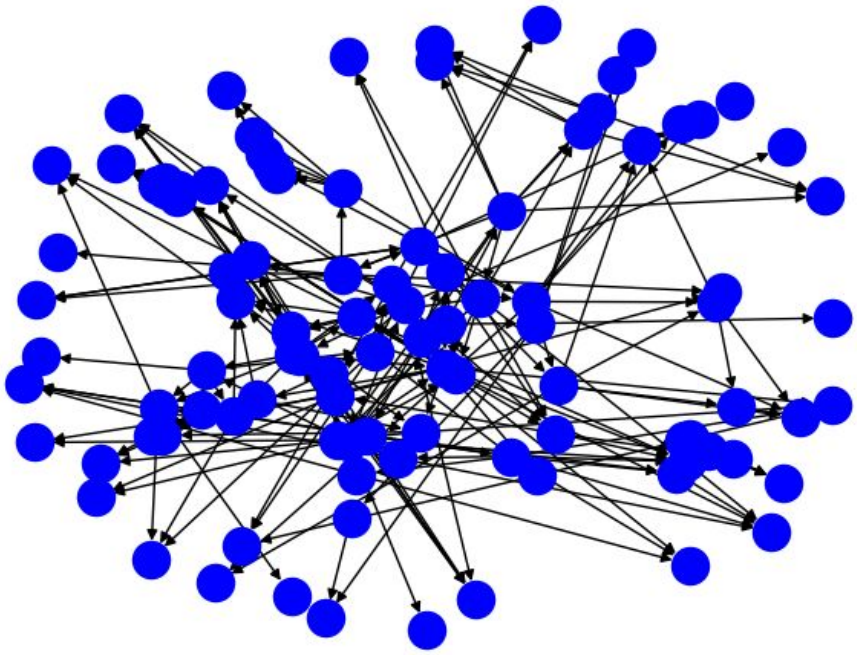

Fig: Subgraph of the Amazon Co-Purchase Dataset created using Networkx Python Library

The network consists of 153439 nodes and 544771 edges. Nodes here represent product. A directed edge between products B to A, represents a co-purchase of B is made when A is bought and vice versa.The graph is a directed graph. Here the edges do not have weight associated with it.

Part3.1 Code for Loading graph

I used NetworkX library's inbuilt module DiGraph object to load network's nodes and edges. I made 2

```
import networkx as nx
Graph = nx.DiGraph()
```

Part3.2 Adding nodes and Edges from file

```
edges = nx.read_edgelist('edges.txt')
nodes = nx.read_adjlist('nodes.txt')

graph.add_edges_from(edges.edges())
graph.add_nodes_from(nodes)
```

Other Visualization

Since the graph was really large, it was not possible to draw the graph. So I took a subgraph and then plotted it using networkx and matplotlib.

```
>> subgraph_nodes = [str(i) for i in range(0, 100)]
>> nx.draw(self.graph.subgraph(subgraph_nodes), node_color='blue', font_size=8, font_weight='bold')
>> plt.savefig("graph.png", format="PNG")
```

Computing Degree of Graph

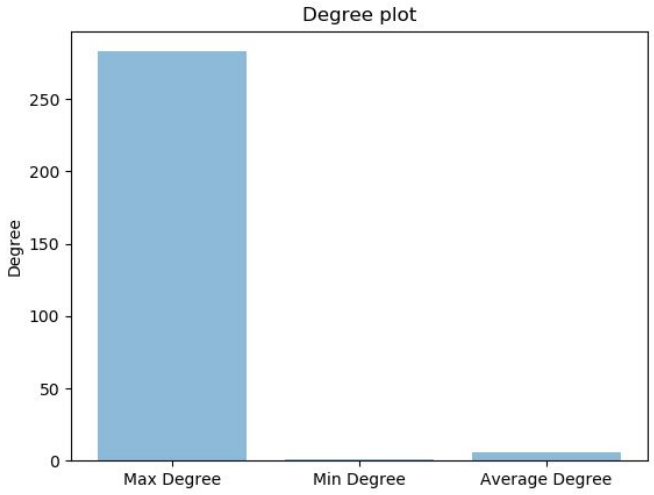

Fig: Degree Plot (Max, Min and Average)

Figure above describes degree properties of the co-purchase network. Here the graph has a maximum degree of 281, minimum degree of 1 and average degree of 6.

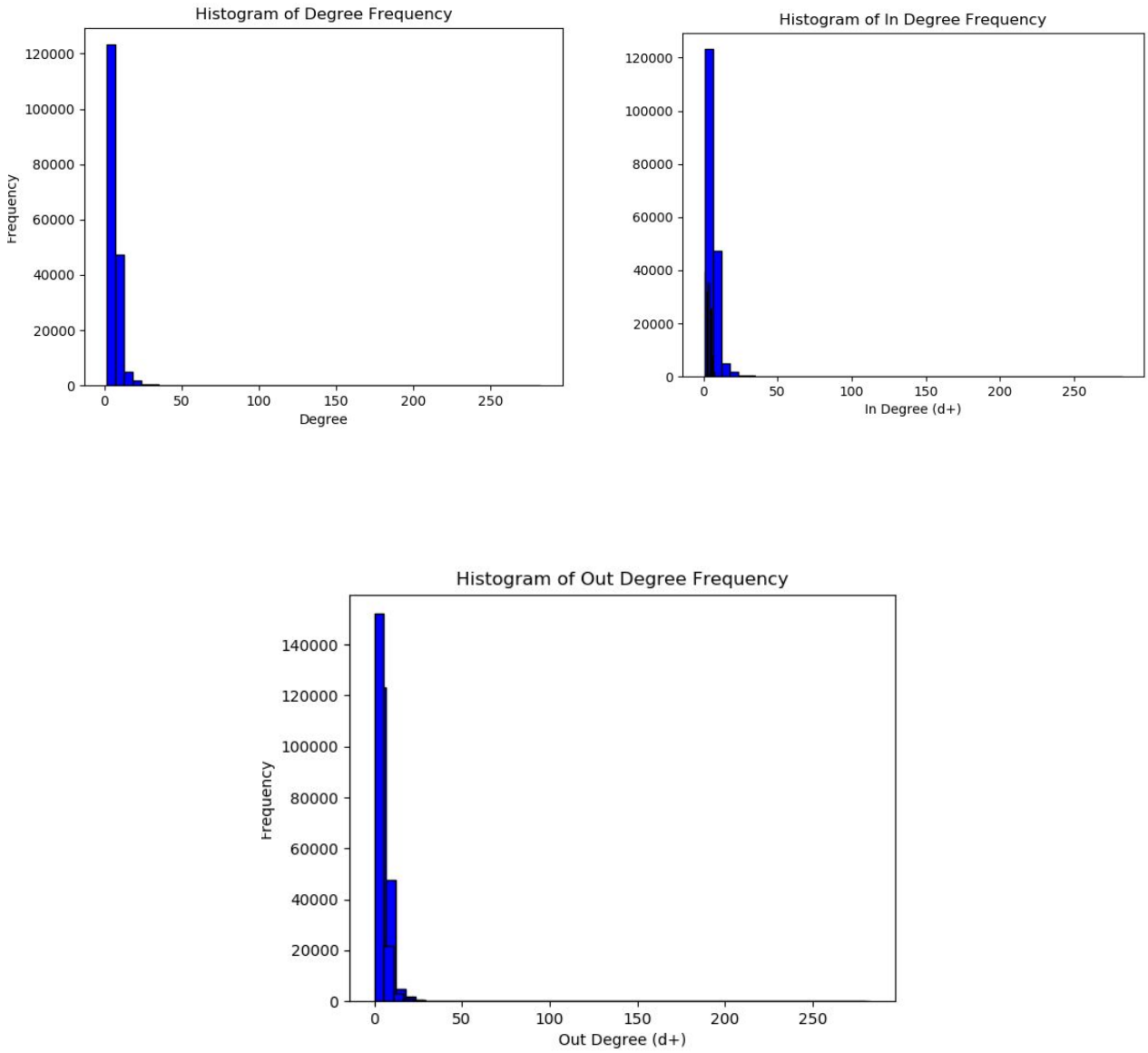

Fig: Degree Distribution plots (Degree, In Degree, Out Degre)

The figure above shows the distribution of different degrees. Indegree of a node in a directed graph is the number of edges pointing towards the node whereas outdegree of a node is the number of nodes pointing away from the node.

Computing Network Centrality

Network centrality is a measure to identify important nodes in the graph or network. In co-purchase dataset of amazon, network centrality can be used to identify key products. using networkx library's inbuilt service to compute centrality. I used in_degree_centrality, out_degree_centrality and degree_centrality functions to compute it.

Code For Computing Network Centrality

```
>> subgraph_nodes = [str(i) for i in range(0, 10000)]
>> centralities = degree_centrality(self.graph.subgraph(subgraph_nodes))
>> centralities_using_degree = sorted(centralities.items(), key=operator.itemgetter(1))

>> in_degree_centralities = in_degree_centrality(self.graph.subgraph(subgraph_nodes))
>> centralities_using_in_degree = sorted(in_degree_centralities.items(),
key=operator.itemgetter(1))

>> out_degree_centralities =
out_degree_centrality(self.graph.subgraph(subgraph_nodes))
>> centralities_using_out_degree = sorted(out_degree_centralities.items(),
key=operator.itemgetter(1))
```

Part 3.3 Traversal (BFS, DFS, RFS) in Co-Purchase Network for Recommendation

In this section, for some random or fixed nodes (products), I tried to traverse the graph using 3 algorithms, i.e BFS, DFS, RFS and compare the time taken to travel for each algorithm. The traversal was limited to certain node or when reached leaves. From all the traversed nodes, the algorithm would recommend those nodes (products) with highest degree. The time required to make the compute the recommendation for different nodes were made and plotted as below.

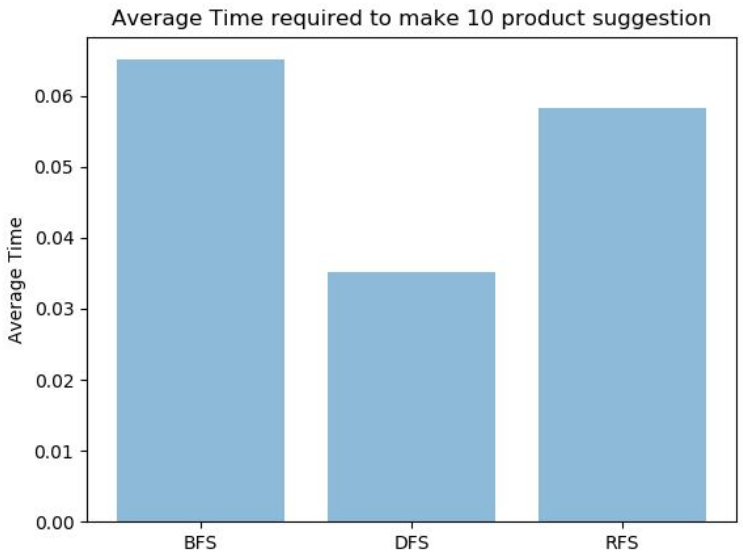

Fig: Comparison of time required for 10 product recommendations using different algorithm

For some fixed points the algorithms were made to crawl the network. Upon crawling from the visited nodes, 10 nodes or products with the highest degree of connectivity were selected and recommended. The time required to make the recommendation was recorded and the graph was plotted as above. We can see, for the same nodes that we took, the average time of traversal and product recommendation was lesser in Depth First Search.

Code for Traversal

```
visited = []
nodes_processed = 0

start = time.time()

Q = self.get_neighbors(startNode)
while (Q):
    nodes_processed += 1
    if nodes_processed > crawlsize:
        break

    # implementing BFS algorithm
    if algorithm == "BFS":
        u = Q[0]
        Q = Q[1:]

    # implement DFS
    elif algorithm == "DFS":
        u = Q.pop()

    # implement RFS
    elif algorithm == "RFS":
        index = random.randint(0, len(Q)-1)
        u = Q.pop(index)

    # add the page to visited list
    visited.append(u)

    # collect all neighbor nodes
    v = self.get_neighbors(u[0])

    for each_v in v:
        if each_v not in Q and each_v not in visited:
            Q.append(each_v)

suggested_products = sorted(visited, key=lambda x: x[1], reverse=True)[0:10]
```

Sample Suggested Products (whole data can be found in traversal.csv)

Start Node	Algorithm	Suggested Products	Time to Suggest
1	BFS	|481||33||8||18||445||6||34||176||2555||210|||	1.404999971
1	DFS	|8||99||837||5||2954||55978||26032||51923||90331||2952|||	0.17412138
1	RFS	|8||18||4656||6||3155||97||719||176||2555||210|||	0.970691442
201	BFS	|1825||899||8645||1389||1824||1930||6488||6578||2050||5056|||	0.25317955

Here, suggested products are nodes of productid.

Part 3.4 Calculating Eigenvalues

For the Larger dataset of 153439 nodes, the computation to find eigen value was taking a long time so I computed eigenvalues using subset of a network of 5,000 nodes. Following is the output:

Largest eigenvalue: 1.7428510473960115
5 Smallest eigenvalues: -3.8289296977614854e-16, 0.14591487809633247, 0.16425214368152083, 0.21441825264702224, 0.2254142296113593

Eigenvalue Graph

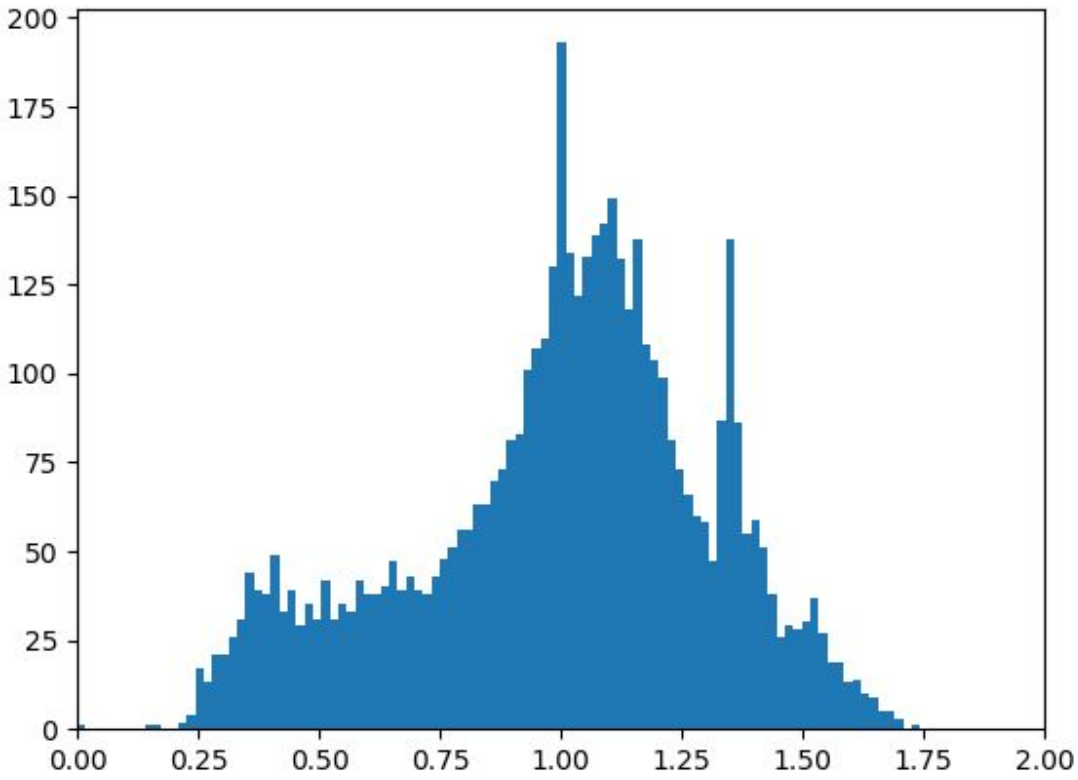

Fig: EigenValues Graph

Tools and Technologies used

I used Python and its library Networkx library for evaluation of network. I also used matplotlib to plot the results of evaluation.

Conclusion

After completing the assignment, I learnt about different types of networks, their properties and different algorithm to perform computation on real world dataset.

References

1. https://snap.stanford.edu/data/#amazon

2. https://snap.stanford.edu/data/amazon0302.html