Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre

COMP 314 (Algorithms and Complexity)

Lab 5 Report

Submitted By:

Mani Dumaru

Roll no.: 15

CE 2019 (3rd year. 2nd semester)

Submitted to:

Dr. Rajani Chulyadyo

Department of Computer Science and Engineering

Submission Date: 4th June, 2023

Python Graph Libraries:

I have used NetworkX python library for this lab work. 6 different networks were downloaded from Network Repository and these networks were imported using the library.

1. Number of Nodes and Edges:

```
def number_of_nodes(graph):
    return len(graph.nodes)

def number_of_edges(graph):
    return len(graph.edges)
```

These are the built in attributes of the graph object from the library that returns the number of nodes and edges it contains.

2. Average degree:

```
def calculate_average_degree(graph):
    sum = 0
    for node in graph.degree:
        sum += node[1]
    avg_degree = sum/len(graph.nodes)
    return avg_degree
```

The for loop in above code calculates the total degree of the network going through the degree of each node. Then the average degree is returned by this function.

3. Density:

```
def calculate_density(graph):
    e = graph.edges
    n = graph.nodes
    if len(n) > 1:
        density = 2*len(e) / (len(n) * (len(n)-1))
    else:
        density = 0
    return density
```

If the number of nodes in the network is greater than 1, its density is calculated by:
density = (2*number of edges) / [number of vertices * (number of vertices - 1)]
else the density is zero.

4. Diameter:

```
def calculate_diameter(graph):
    if not nx.is_connected(graph):
        return "infinite"
    diameter = 0
    nodes = graph.nodes
    for i in nodes:
        for j in nodes:
            if i == j:
                continue
            path_length = nx.shortest_path_length(graph, source = i, target = j)
            if path_length > diameter:
                diameter = path_length
    return diameter
```

Diameter of a network is the shortest path between the two most distanced nodes. For a graph that is not connected (i.e.: there is not path to travel between any two vertices), the diameter is infinite.

i. Going through each node, find the shortest path to all the other nodes.
ii. Choose the largest path among these paths as the diameter.
iii. Loop through each and every vertex and keep on replacing largest paths of larger size than the current diameter.
iv. After the completion of the loop, largest of such path is the final diameter of the network.

For this lab, calculation of diameter for networks with around 5K vertices took very long time. So, I've commented the diameter calculation portion.

5. Clustering Coefficient:

```
def calculate_clustering(graph):
    nodes = graph.nodes
    total_cluster = 0
    for i in nodes:
        neighbour = list(nx.neighbors(graph, i))
        connection = 0
        for j in neighbour:
            for k in neighbour:
                if nx.is_path(graph, [j,k]):
                    connection += 1

        # connection should be fivided by 2 here because
        # the loop counts 1,2 as a connection and also counts
        # 2,1 as another connection since it is an undirected graph.

        if len(neighbour) > 1:
            total_cluster += (2*(connection/2))/ (len(neighbour) * (len(neighbour)-1))

    c = total_cluster / len(nodes)
    return c
```

There is a built-in function in the library that returns all the neighbours of a particular node. Now to find the connections between the neighbours of a particular node, each

neighbour is looped through each other and if there exists a path between them, the connection is incremented. In this way the connection and total number of neighbours is calculated and hence using these values, clustering coefficient can be calculated.

6. Degree distribution:

```
def degree_distribution(graph):
    degree = graph.degree
    max_degree = max(y for (x,y) in degree)
    c_degree = [0 for _ in range(0, max_degree+1)]
    for i in range(0, max_degree+1):
        count = 0
        for (x,y) in degree:
            if y == i:
                count += 1
        if i == 0:
            pk = 0
        else:
            pk = count/len(graph.nodes)

        c_degree[i] = pk
    return c_degree
```

From all the degrees of the network, the maximum degree is found. For each degree from 0 to maximum degree, total number of times that particular degree is repeated is calculated and is divided by total number of nodes in the graph. This gives the degree distribution P(k). These values are then plotted against the degrees itself to form degree distribution graph.

Outputs:

For Task 1:

A small graph with only 46 nodes:

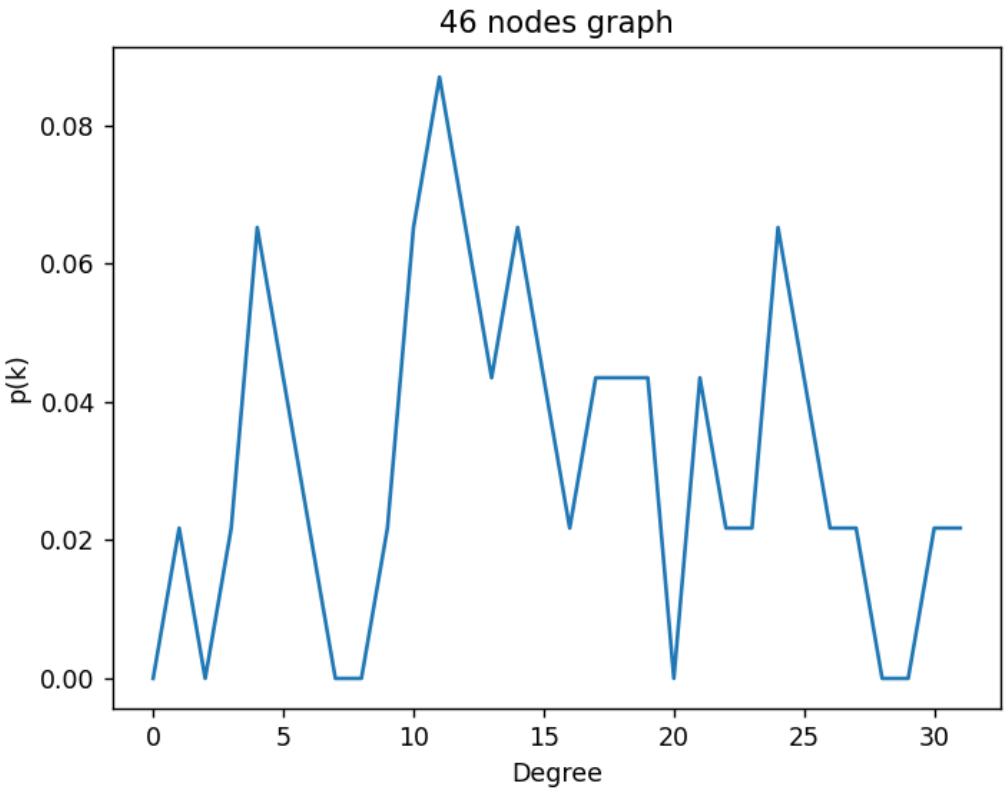
```
D:\CE-2019\Sem 6\lab works\Algorithm\lab5>python small.py
Number of Nodes = 46
Number of Edges = 348

Average degree = 15.130434782608695

calculated density = 0.336231884057971
built in function density = 0.336231884057971

calculated diameter = 4
built in diameter = 4

calculated clustering coefficient = 0.6877074078508906
built in function clustering = 0.6877074078508906
```



For Task 2:

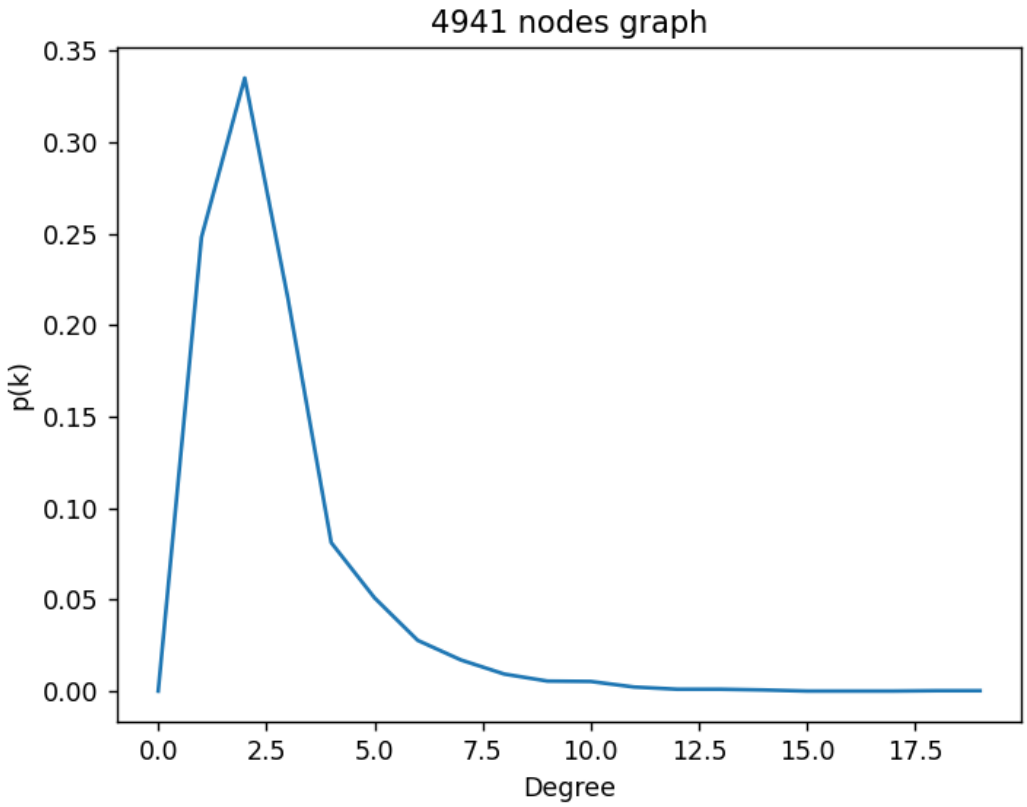
1. inf-power.mtx

```
Number of Nodes = 4941
Number of Edges = 6594

Average degree = 2.66909532483303

calculated density = 0.0005403026973346214
built in function density = 0.0005403026973346214

calculated clustering coefficient = 0.08010361108159714
built in function clustering = 0.08010361108159714
```



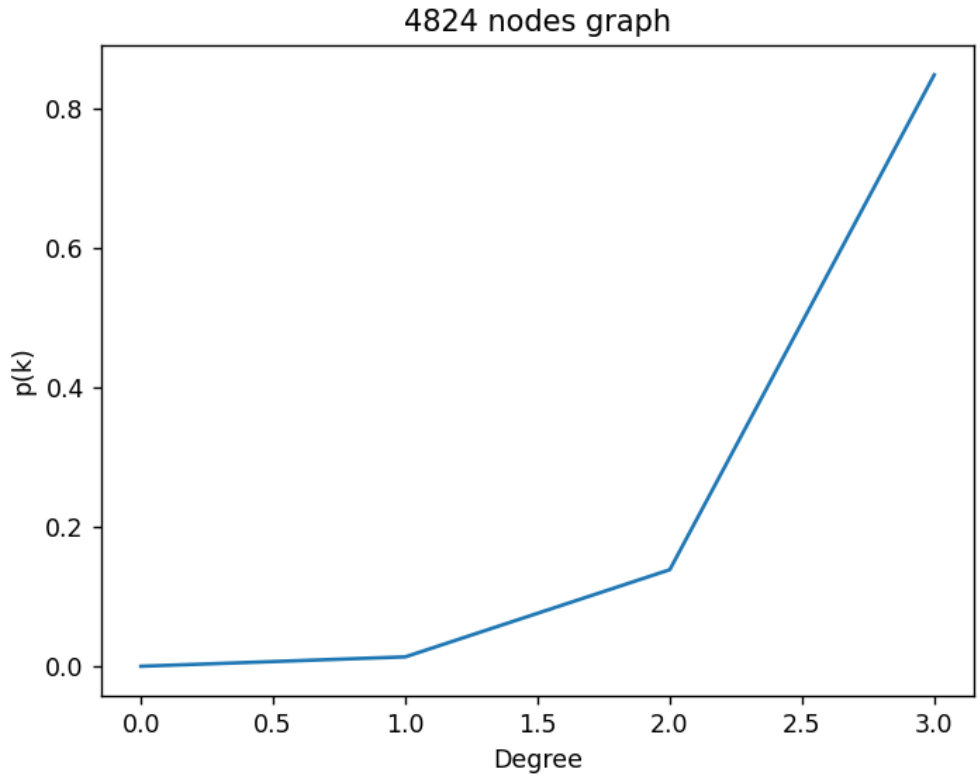
2. uk.mtx

```
------------------------------------------------------------
Number of Nodes = 4824
Number of Edges = 6837

Average degree = 2.834577114427861

calculated density = 0.0005877207369744683
built in function density = 0.0005877207369744683

calculated clustering coefficient = 0.00020729684908789387
built in function clustering = 0.00020729684908789387
```



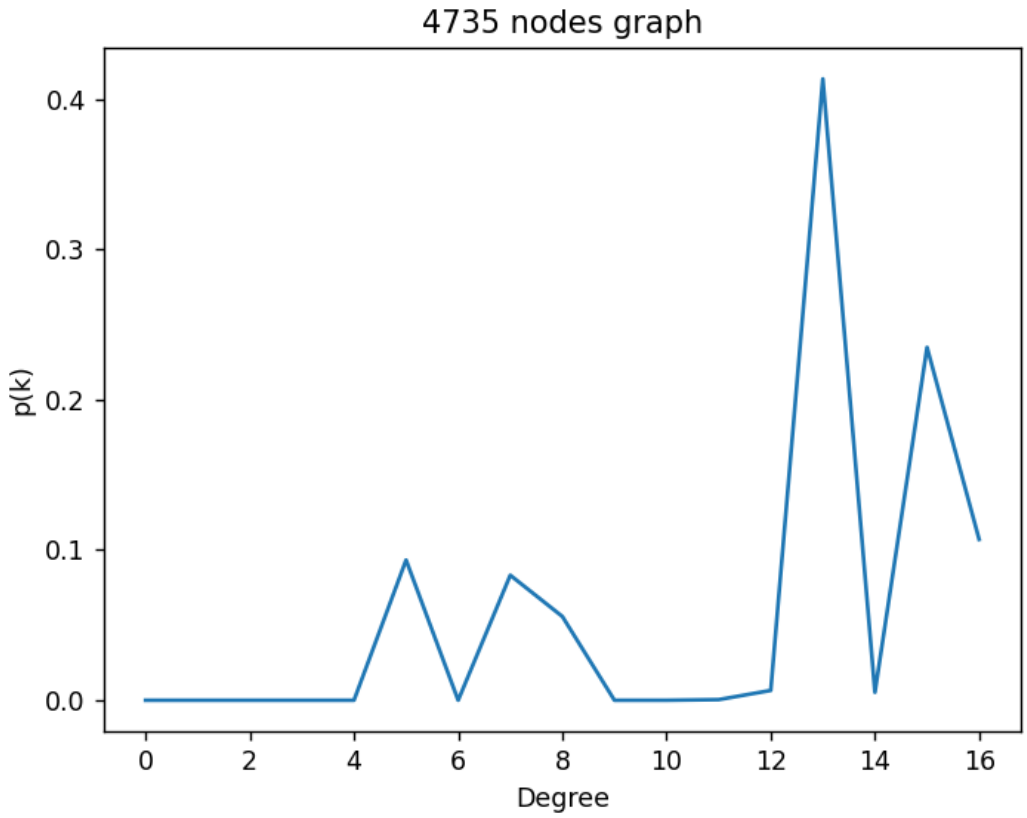
3. n4c5-b7.mtx

```
------------------------------------------------------------
Number of Nodes = 4735
Number of Edges = 29035

Average degree = 12.263991552270328

calculated density = 0.002590619254809955
built in function density = 0.002590619254809955

calculated clustering coefficient = 0.0010263179270571007
built in function clustering = 0.0009793216287408448
```



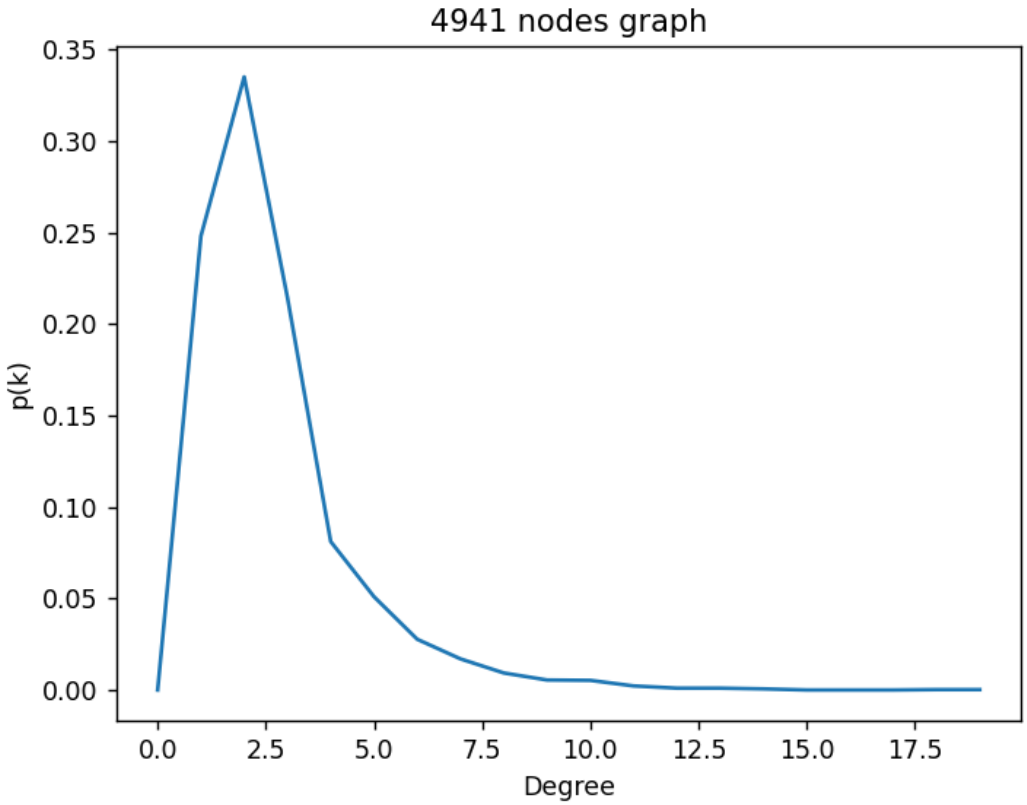
4. USpowerGrid.mtx

```
Number of Nodes = 4941
Number of Edges = 6594

Average degree = 2.66909532483303

calculated density = 0.0005403026973346214
built in function density = 0.0005403026973346214

calculated clustering coefficient = 0.08010361108159714
built in function clustering = 0.08010361108159714
```



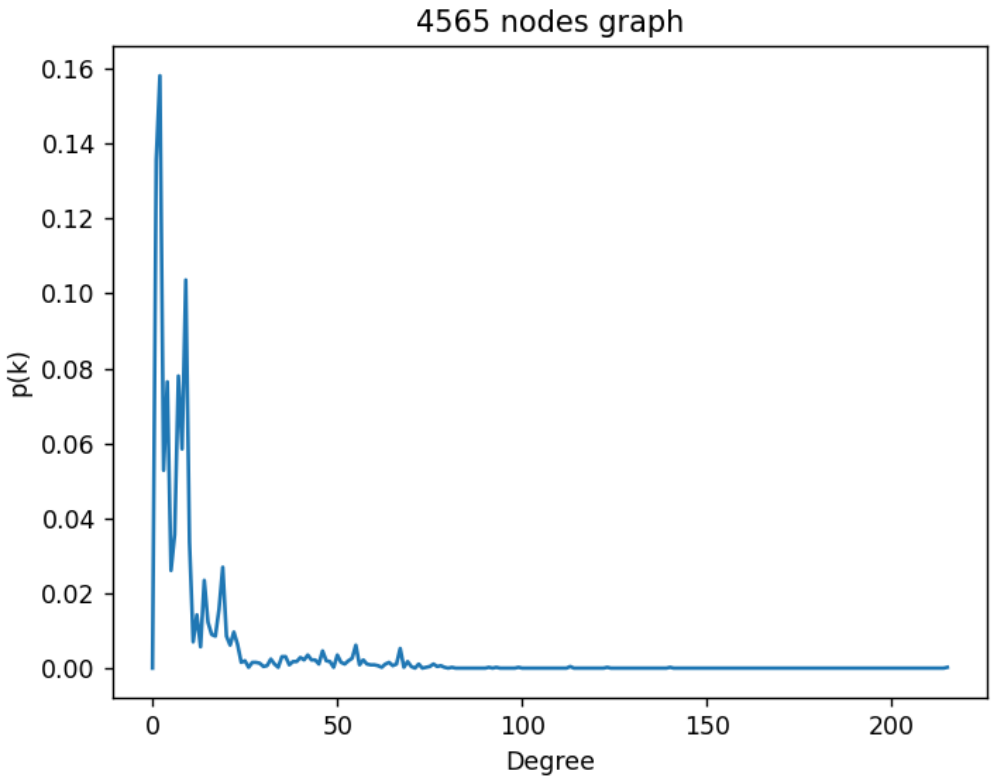
5. model3.mtx

```
Number of Nodes = 4565
Number of Edges = 23972

Average degree = 10.50251916757941

calculated density = 0.0023011654617833936
built in function density = 0.0023011654617833936

calculated clustering coefficient = 0.0009177543307261174
built in function clustering = 0.0009177543307261174
```



Answer the following Questions:

1. From the network properties, what can you say about the networks you selected?
   From the networks that I selected, the diameters of networks with many numbers of nodes could not be found. It took a very long amount of time to calculate the diameter of such networks since during the calculation, each node must be visited and its shortest path to all other nodes must be calculated. For this to be done for more than 5 thousand nodes of the graph, it took way too long. Also, the diameter of disconnected graph is infinite since there is no path between two of the nodes. Hence their distance being infinite.

2. Did you find any patterns in the degree of distribution of the networks? In any case, can you come to ay conclusion about the graph from their degree distribution?
   For three of the outcomes, lower degree has higher frequency while the increase in degree brings their frequency lower. In the other two cases, higher degree is frequently repeated while as the degree gets lower its frequency also gets lower. Lower degree appears less frequently in such networks. It the frequency of higher degrees are evenly distributed; we can conclude that the graph is dense in nature. While the frequency of lower degree is hight means that the graph is sparse.