BTS | Barcelona Technology School

Complex and Social Networks

Advanced Data Analysis
Master in Big Data Solutions 2020-2021

Ankit Tewari
Chief Instructor, Advanced Data Analysis
ankit.tewari@bts.tech

Contents

- Understanding Networks: Making sense of various types of networks
- Understanding Network Analysis: Exploring fundamental building blocks of a network and how they support analysis
- Network Representation
- Degree Distribution
- Advanced Network Visualisation Methods
- Travelling through the Network
- Influence Estimation
- Clustering in Networks

BTS | Barcelona Technology School

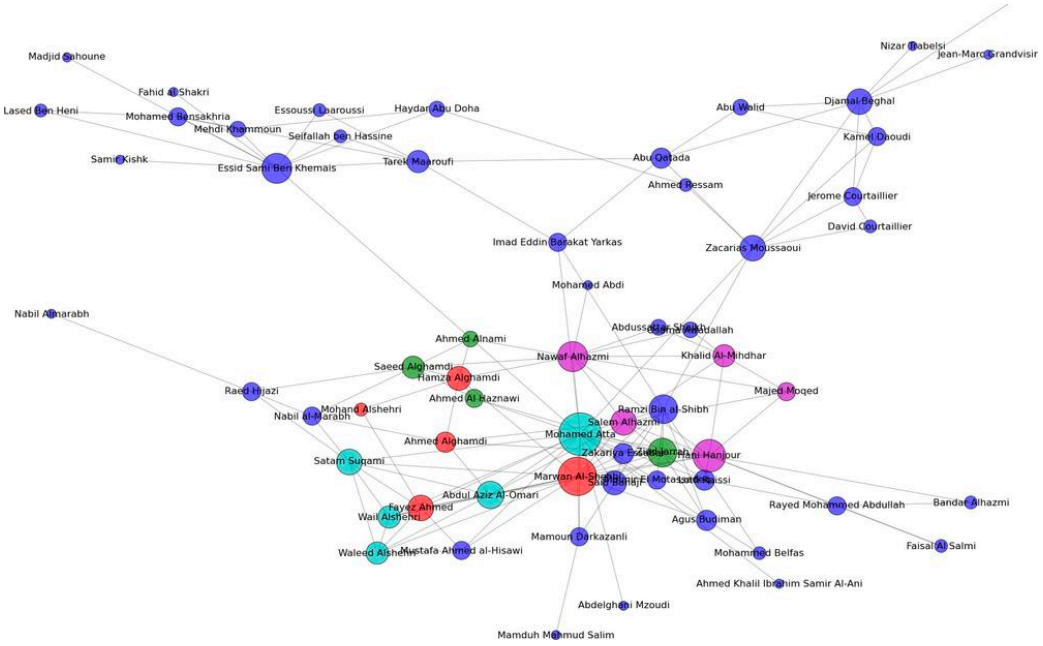
Understanding Networks

Introduction to Networks

Social Networks

Social Networks are usually the networks in which the nodes denote the individuals or persons and the link between the nodes denote the interactions (such as friendship, collaboration etc.) between the individuals.

The example below is an example of Social Network of Hamburg Cell (which according to the United States started the planning for 9/11 and ultimately participated in the attack).



According to Marc Sageman, after studying the lives of 172 members of the Hamburg Cell, it was found "the most common factor driving them was the social ties within their cell with most starting as friends, colleagues, or relatives and were drawn closer by bonds of friendship, loyalty, solidarity and trust, and rewarded by a powerful sense of belonging and collective identity".

Remark: The facts stated above in no way reflect the opinion of the author but merely are based on the data available.

Introduction to Networks

Infrastructure Networks

Infrastructure Networks are usually the networks in which the nodes denote the stations or stops and the link between the nodes denote the mode of transport (such as metro rail, tram service etc.) between the nodes.

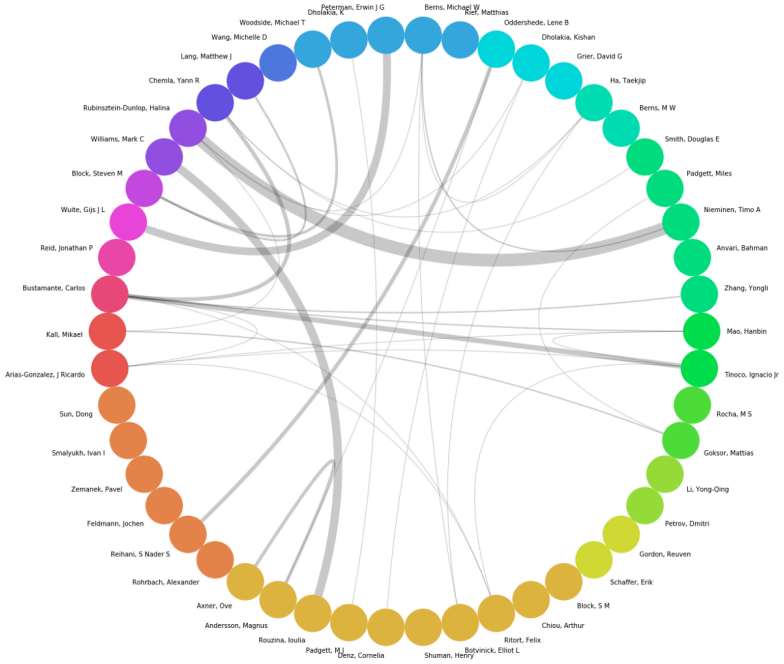
The example below is an example of Transport Network of Barcelona (which includes various modes of transport such as Trams, Sub Urban Trains etc.).

Introduction to Networks

Information Networks

Information Networks are usually the networks in which the nodes generally denote the entities responsible for origination or assimilation of information such as researchers or institutions and the link between the nodes denote the interactions (such as academic collaboration or grants or active information exchange etc.) between the entities.

The example below is an example of Academic Citation Network of various researchers (which can be used to determine eventually how strong collaboration or citation history an author have).

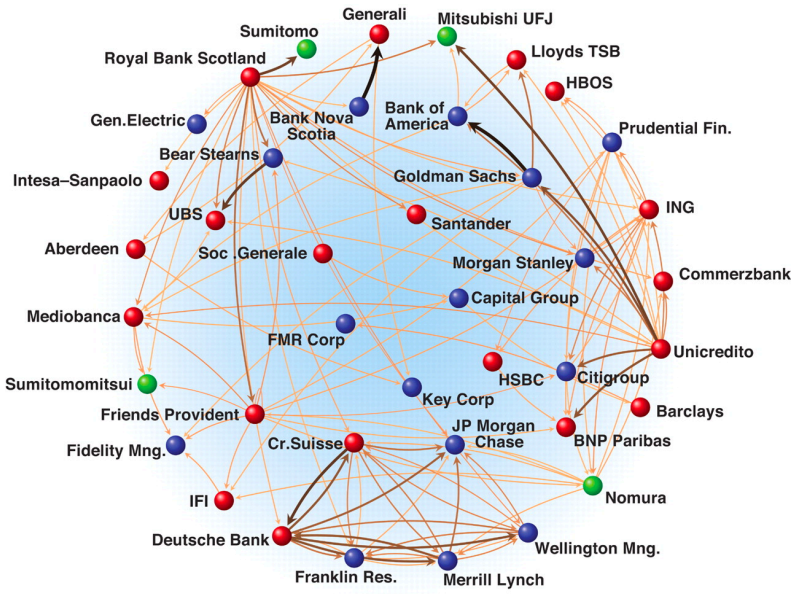


Introduction to Networks

Financial Networks

Financial Networks are usually the networks in which the nodes denote the financial entities such as banks or governments and the link between the nodes denote the transactions (such as fund transfers, currency exchange rate etc.) between the entities.

The example below is an example of Global Network of Banks.

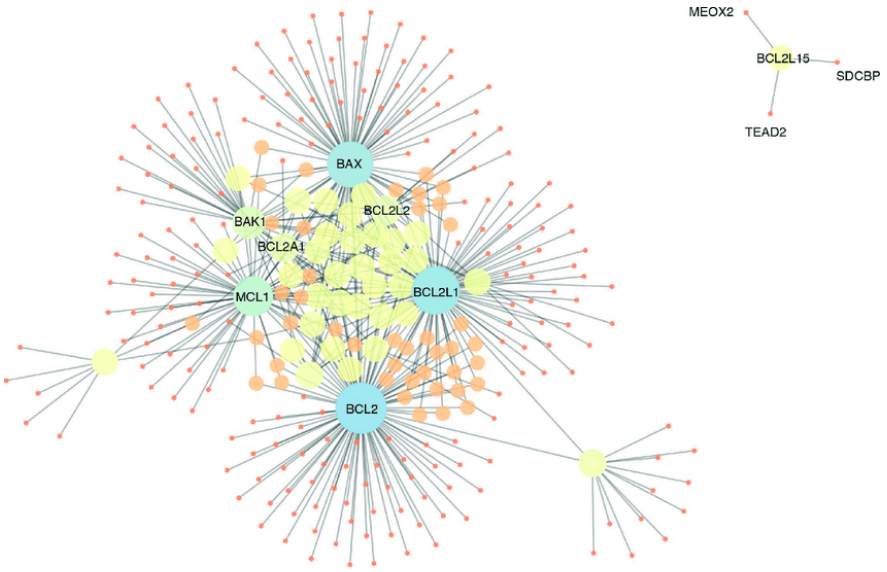


Introduction to Networks

Biological Networks

Biological Networks are usually the networks in which the nodes denote the individuals biological agents or entities and the link between the nodes denote the interactions (such as protein-protein interaction etc.) between the entities.

The example below is an example of Biological Network (which describes how various proteins and associated and interacted with each other).

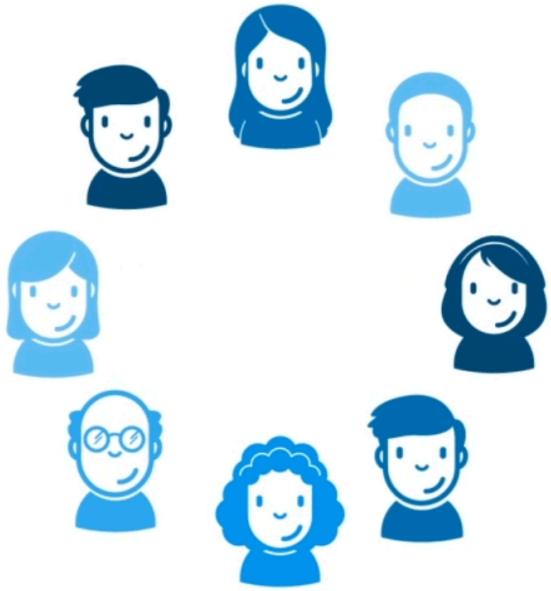


BTS | Barcelona Technology School

What constitutes a network? Components?

Components of Networks

Objects: Nodes or Vertices



Components of Networks

Interaction: Links or Edges

Components of Networks

System: Network or Graphs

Network Analysis

What is Network Analysis?

Network analysis is the process of making sense of complex relationships that exists inside a network and represented by means of its components.

Therefore, network analysis is a term that may be broadly referring to one or more activities of the following type:

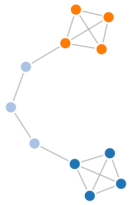
1. Influence Estimation: Identifying the most influential entities (nodes) in a network
2. Pathfinding: Exploring the fastest options to travel from one end of the network to another
3. Clustering: Detecting the existence of any sub-groups amongst the entities (nodes) of the network

Network Analysis

Network Analysis in Python

NetworkX
Network Analysis in Python

NetworkX is a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks.



Software for complex networks

- Data structures for graphs, digraphs, and multigraphs
- Many standard graph algorithms
- Network structure and analysis measures
- Generators for classic graphs, random graphs, and synthetic networks
- Nodes can be "anything" (e.g., text, images, XML records)
- Edges can hold arbitrary data (e.g., weights, time-series)
- Open source 3-clause BSD license
- Well tested with over 90% code coverage
- Additional benefits from Python include fast prototyping, easy to teach, and multi-platform

What the the fundamental building blocks of a network in NetworkX?

- Instantiated Graph Object
- Graph Nodes
- Graph Edges

Once we have the availability of these fundamental structures, we can create a graph in NetworkX and represent our network.

Let us understand how.

Network Analysis

Getting Started: Assigning Nodes

Network analysis is the process of making sense of complex relationships that exists inside a network and represented by means of its components.

Let us begin with creating our first network. It will consist of nodes- A, B, C, D and, E;
Further, the nodes are connected together by means of edges- AB, AC, BC, BD, CE and, DE;

```
import networkx as nx

G = nx.Graph()
nodes= ["A", "B", "C", "D", "E"]
G.add_nodes_from(nodes)
print("Object type of G: ", type(G))
print("Nodes of Graph G", G.nodes)
```

```
Object type of G:  <class 'networkx.classes.graph.Graph'>
Nodes of Graph G ['A', 'B', 'C', 'D', 'E']
```

Network Analysis

Getting Started: Assigning Edges

Network analysis is the process of making sense of complex relationships that exists inside a network and represented by means of its components.

Let us begin with creating our first network. It will consist of nodes- A, B, C, D and, E;
Further, the nodes are connected together by means of edges- AB, AC, BC, BD, CE and, DE;

```
import networkx as nx

G = nx.Graph()
nodes= ["A", "B", "C", "D", "E"]
G.add_nodes_from(nodes)
print("Object type of G: ", type(G))
print("Nodes of Graph G", G.nodes)
```

```
Object type of G:  <class 'networkx.classes.graph.Graph'>
Nodes of Graph G ['A', 'B', 'C', 'D', 'E']
```

```
edges= [("A", "B"), ("A", "C"), ("B", "C"), ("C", "E"), ("B", "D"), ("D", "E")]
G.add_edges_from(edges)
print("Edges of Graph G:", G.edges)
```

```
Edges of Graph G: [('A', 'B'), ('A', 'C'), ('B', 'C'), ('B', 'D'), ('C', 'E'), ('D', 'E')]
```

Network Analysis

Getting Started: Assigning Metadata

Network analysis is the process of making sense of complex relationships that exists inside a network and represented by means of its components.

Let us begin with creating our first network. It will consist of nodes- A, B, C, D and, E;
Further, the nodes are connected together by means of edges- AB, AC, BC, BD, CE and, DE;

```
import networkx as nx

G = nx.Graph()
nodes= ["A", "B", "C", "D", "E"]
G.add_nodes_from(nodes)
print("Object type of G: ", type(G))
print("Nodes of Graph G", G.nodes)
```

```
Object type of G:  <class 'networkx.classes.graph.Graph'>
Nodes of Graph G ['A', 'B', 'C', 'D', 'E']
```

```
edges= [("A", "B"), ("A", "C"), ("B", "C"), ("C", "E"), ("B", "D"), ("D", "E")]
G.add_edges_from(edges)
print("Edges of Graph G:", G.edges)
```

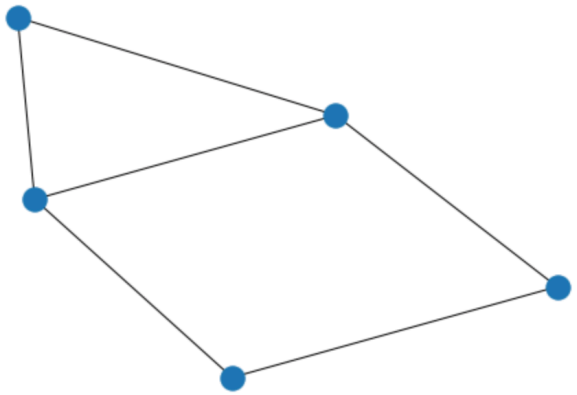
```
Edges of Graph G: [('A', 'B'), ('A', 'C'), ('B', 'C'), ('B', 'D'), ('C', 'E'), ('D', 'E')]
```

```
G.nodes["A"]['label'] = 'Alpha'
G.nodes["A"]['description'] = 'First member of the team'
G.nodes["A"]['color'] = 'Red'
print(G.nodes(data= True))
```

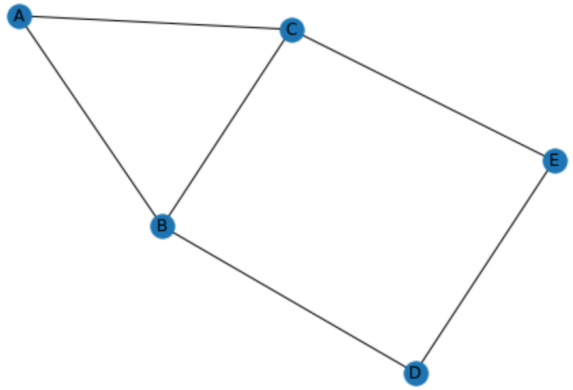
Network Analysis

Getting Started: Visualizing Networks

```
import matplotlib.pyplot as plt
nx.draw(G)
plt.show()
```



```
nx.draw(G, with_labels= True)
```



BTS | Barcelona Technology School

Graph Representation

How do we represent networks?

Network Representation

Adjacency Matrix

Networks are represented as Graphs. Graphs are mathematical data structures that consist of the same components as that of the network i.e. nodes and edges.

Representing a network fundamentally means storing the Graph which is representing the network in the memory of the computer.

Adjacency matrices are one of the most common ways to represent the graphs. In a typical adjacency matrix, we basically exploit the fact that each of the nodes can be represented as indices of the matrix and accordingly, each relation between the nodes is represented as an element of the matrix.

Network Representation

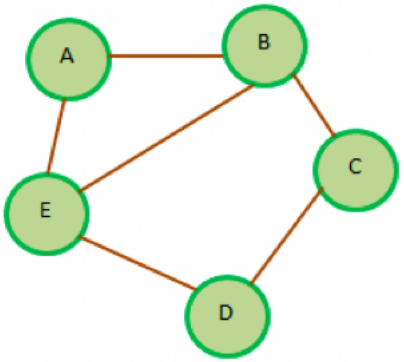
Adjacency Matrix

Networks are represented as Graphs. Graphs are mathematical data structures that consist of the same components as that of the network i.e. nodes and edges.

Representing a network fundamentally means storing the Graph which is representing the network in the memory of the computer.

Adjacency matrices are one of the most common ways to represent the graphs. In a typical adjacency matrix, we basically exploit the fact that each of the nodes can be represented as indices of the matrix and accordingly, each relation between the nodes is represented as an element of the matrix.

For example, consider the following graph:



	A	B	C	D	E
A	0	1	0	0	1
B	1	0	1	0	1
C	0	1	0	1	0
D	0	0	1	0	1
E	1	1	0	1	0

Nodes of the Graph, represented as indices of the matrix

Relations among the nodes (edges) represented as an element of the matrix

Network Representation

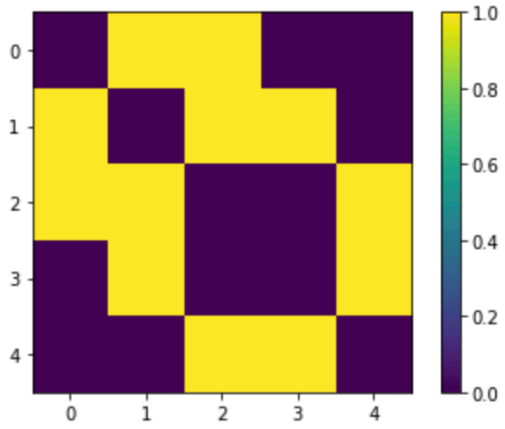
Adjacency Matrix

For example, consider the following code corresponding to the graph we had witnessed before:

```
nx.adjacency_matrix(G).todense()
```

```
matrix([[0, 1, 1, 0, 0],
        [1, 0, 1, 1, 0],
        [1, 1, 0, 0, 1],
        [0, 1, 0, 0, 1],
        [0, 0, 1, 1, 0]])
```

```
import numpy as np
plt.imshow(nx.adjacency_matrix(G).todense())
plt.colorbar()
plt.show()
```



Network Representation

How about considering a real world data?

Networks are represented as Graphs. Graphs are mathematical data structures that consist of the same components as that of the network i.e. nodes and edges.

```
import networkx as nx
from pandas import read_csv
routes= read_csv("routes.csv")
routes.head(5)
```

	Airline	Airline ID	Source airport	Source airport ID	Destination airport	Destination airport ID	Codeshare	Stops	Equipment
0	2B	410	ASF	2966	KZN	2990	NaN	0	CR2
1	2B	410	ASF	2966	MRV	2962	NaN	0	CR2
2	2B	410	CEK	2968	KZN	2990	NaN	0	CR2
3	2B	410	CEK	2968	OVB	4078	NaN	0	CR2
4	2B	410	DME	4029	KZN	2990	NaN	0	CR2

Network Representation

Converting tabular data into a Network

Networks are represented as Graphs. Graphs are mathematical data structures that consist of the same components as that of the network i.e. nodes and edges.

```
import networkx as nx
from pandas import read_csv
routes= read_csv("routes.csv")
routes.head(5)
```

	Airline	Airline ID	Source airport	Source airport ID	Destination airport	Destination airport ID	Codeshare	Stops	Equipment
0	2B	410	ASF	2966	KZN	2990	NaN	0	CR2
1	2B	410	ASF	2966	MRV	2962	NaN	0	CR2
2	2B	410	CEK	2968	KZN	2990	NaN	0	CR2
3	2B	410	CEK	2968	OVB	4078	NaN	0	CR2
4	2B	410	DME	4029	KZN	2990	NaN	0	CR2

```
nodes= [row["Source airport"] for item, row in routes.iterrows()]
edges= [(row["Source airport"], row["Destination airport"]) for item, row in routes.iterrows()]
```

```
routes_graph= nx.Graph()
routes_graph.add_nodes_from(nodes)
routes_graph.add_edges_from(edges)
```

BTS | Barcelona Technology School

Degree Distribution

Degree Distribution

Understanding Degree Distribution

Consider that we have some Graph G, such that,

We have,
N: finite number of vertices corresponding to the graph (total number of vertices in the Graph G)

And,
M(K): number of vertices of degree K (for example, M(2) means number of vertices of degrees 2)

Therefore, we define,
M(1), M(2),...,M(N) as the degree spectrum (loops are allowed).

Accordingly,
M(K) / N: the proportion of vertices of degree K, which defines the (empirical) degree distribution.

Therefore,
P(K): function giving the probability that a vertex has degree K,

Probability Mass Function: $P(K) \approx N(K)/N$

Read more about the degree distribution here

Degree Distribution

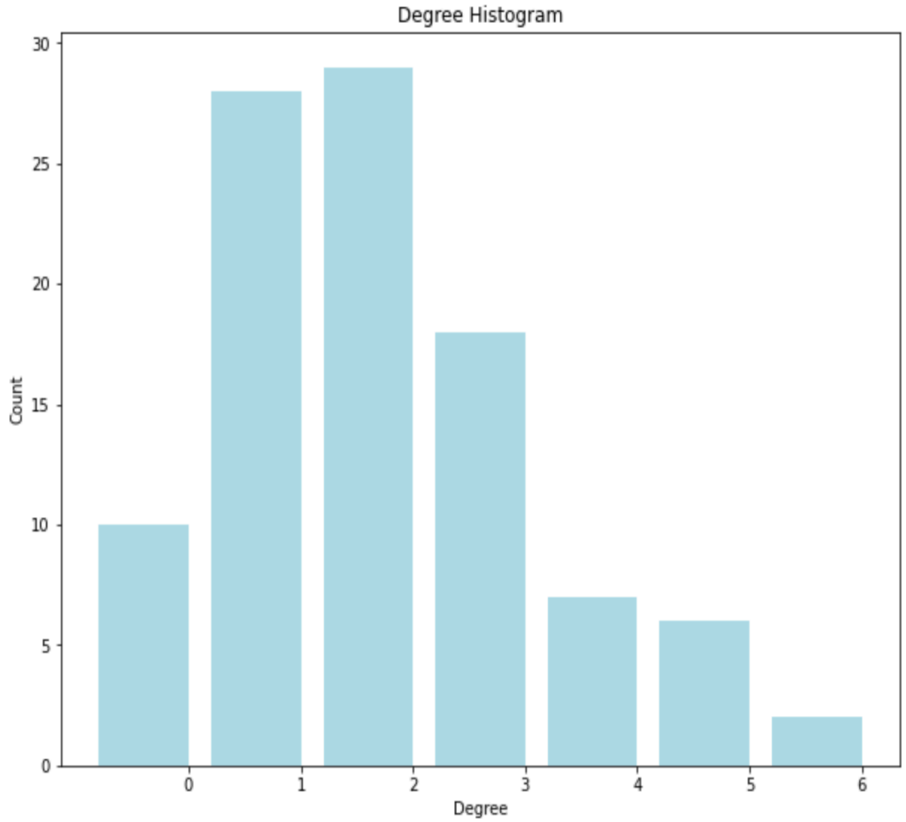
Random Graphs

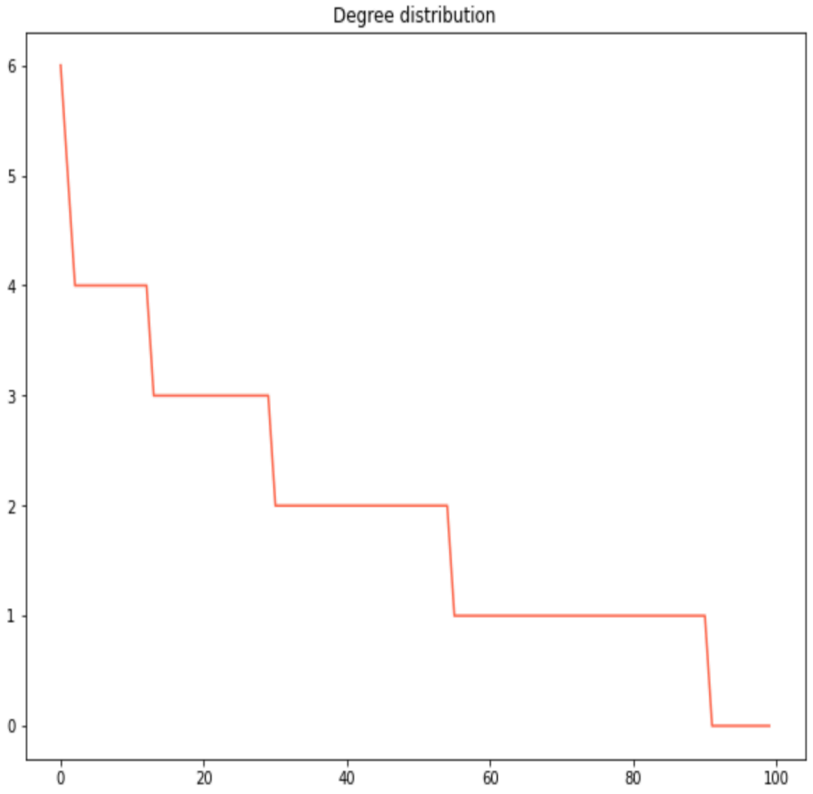
```
import collections
import matplotlib.pyplot as plt


random_graph = nx.gnp_random_graph(100, 0.02)

degree_sequence = sorted([d for n, d in random_graph.degree()], reverse=True)
degreeCount = collections.Counter(degree_sequence)
deg, cnt = zip(*degreeCount.items())

fig, ax = plt.subplots()
plt.bar(deg, cnt, width=0.80, color="lightblue")

plt.title("Degree Histogram")
plt.ylabel("Count")
plt.xlabel("Degree")
ax.set_xticks([d + 0.4 for d in deg])
ax.set_xticklabels(deg)
```

```
degree_distribution= [int(degree[1]) for degree in random_graph.degree]
degree_distribution.sort(reverse= True)
plt.rcParams['figure.figsize'] = (10.0, 8.0)
plt.plot(degree_distribution, color= "tomato")
plt.title("Degree distribution")
```

Degree Distribution

Random Graphs





Degree Distribution

Real World Graphs

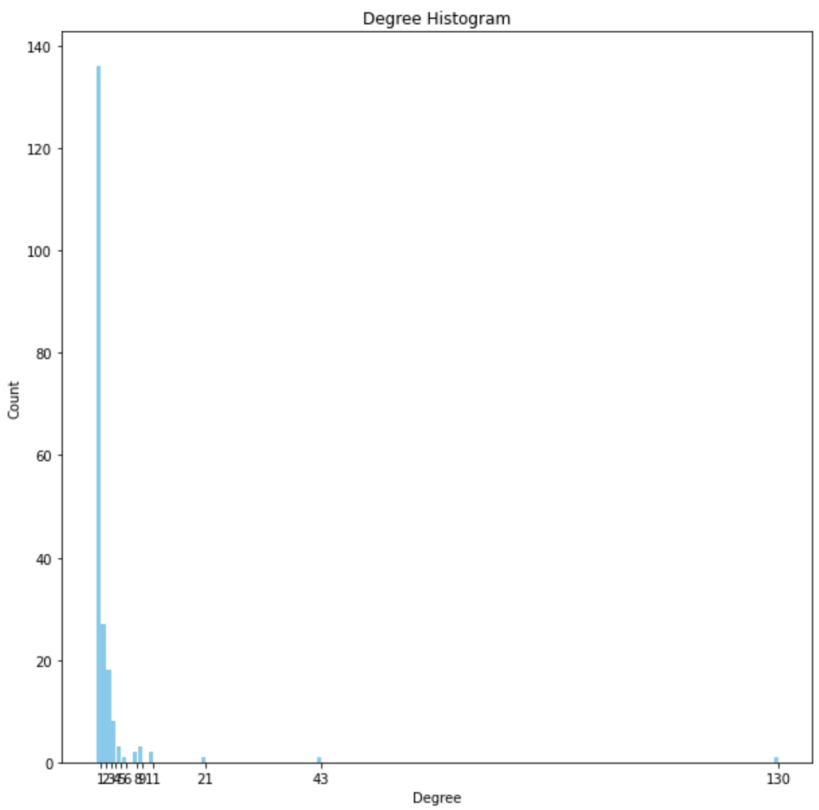
```
import collections
import matplotlib.pyplot as plt

degree_sequence = sorted([d for n, d in british_routes.degree()], reverse=True)
degreeCount = collections.Counter(degree_sequence)
deg, cnt = zip(*degreeCount.items())

fig, ax = plt.subplots()
plt.bar(deg, cnt, width=0.80, color="lightblue")

plt.title("Degree Histogram")
plt.ylabel("Count")
plt.xlabel("Degree")
ax.set_xticks([d + 0.4 for d in deg])
ax.set_xticklabels(deg)
```

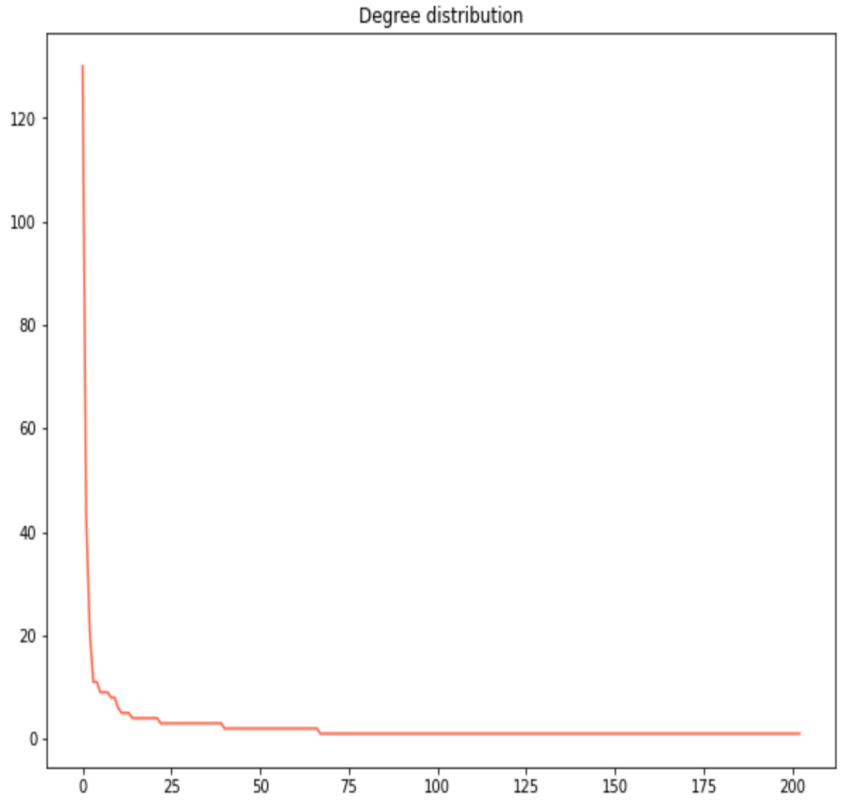
```
degree_distribution= [int(degree[1]) for degree in british_routes.degree]
degree_distribution.sort(reverse= True)
plt.rcParams['figure.figsize'] = (10.0, 8.0)
plt.plot(degree_distribution, color= "tomato")
plt.title("Degree distribution")
```

Degree Distribution

Real World Graphs

The real world graphs resemble a Power Law. Observe the two figures below and draw inferences;

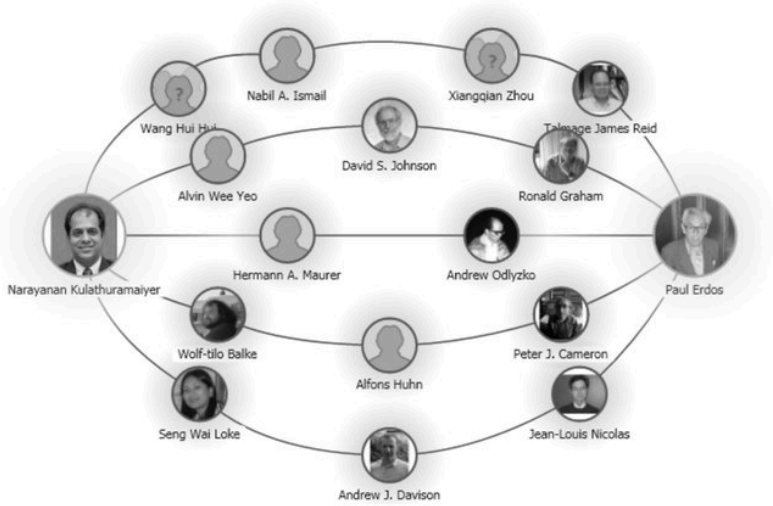




Small World Phenomenon

Small World Phenomenon

Social networks are so rich in short paths that it is known as the small-world phenomenon, or the "six degrees of separation" and it has long been the subject of both anecdotal and scientific fascination. The first significant empirical study of the small-world phenomenon was undertaken by the social psychologist Stanley Milgram, who asked randomly chosen "starter" individuals to each try forwarding a letter to a designated "target" person living in the town of Sharon, MA, a suburb of Boston.



He provided the target's name, address, occupation, and some personal information, but stipulated that the participants could not mail the letter directly to the target;

Rather, each participant could only advance the letter by forwarding it to a single acquaintance that he or she knew on a first-name basis, with the goal of reaching the target as rapidly as possible. Roughly a third of the letters eventually arrived at the target, in a median of six steps and this has since served as basic experimental evidence for the existence of short paths in the global friendship network, linking all (or almost all) of us together in society. This style of experiment, constructing paths through social networks to distant target people, has been repeated by a number of other groups in subsequent decades.

Read more about the small world phenomenon here

BTS | Barcelona Technology School

Graph Visualization

Network Representation

Finding Busiest Airports Visually

Suppose our objective of this study is to find which airports are the busiest in a network of airports served by British Airways.

```
import matplotlib.pyplot as plt
import seaborn as sns

british_routes= routes[routes["Airline"]== "BA"]
nodes= [row["Source airport"] for item, row in british_routes.iterrows()]
edges= [(row["Source airport"], row["Destination airport"]) for item, row in british_routes.iterrows()]
british_routes= nx.Graph()
british_routes.add_nodes_from(nodes)
british_routes.add_edges_from(edges)
```

Network Representation

Finding Busiest Airports Visually

Suppose our objective of this study is to find which airports are the busiest in a network of airports served by British Airways.

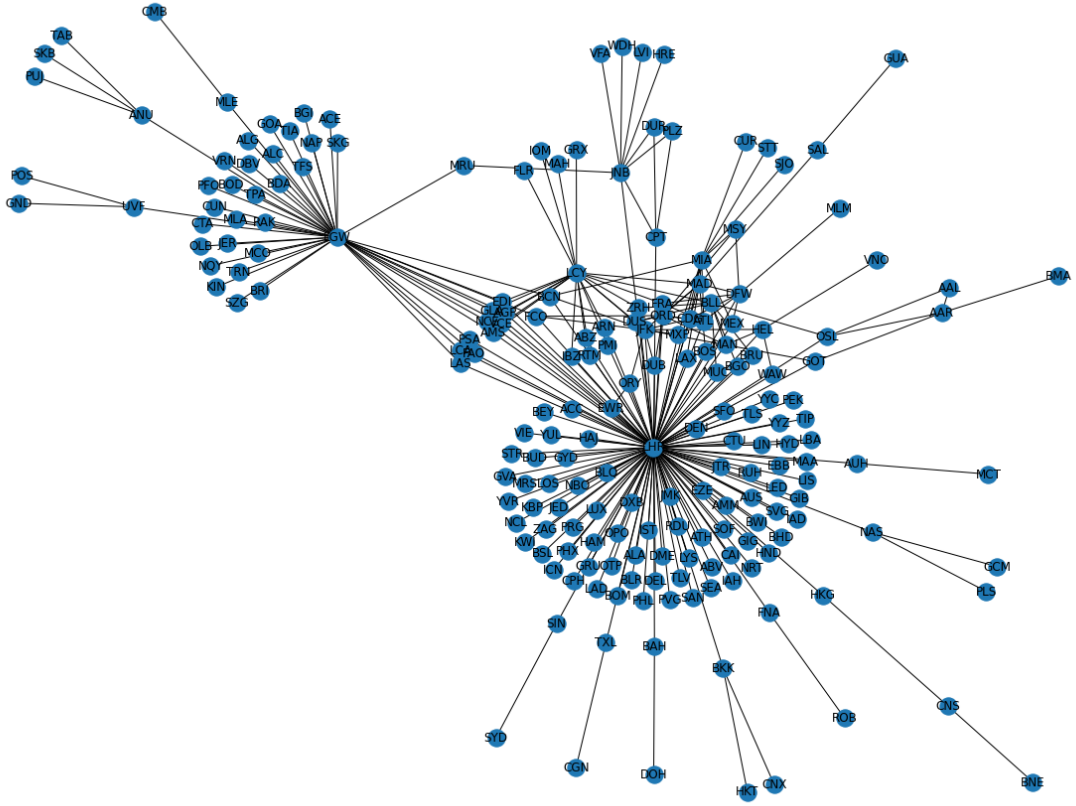
```
import matplotlib.pyplot as plt
import seaborn as sns

british_routes= routes[routes["Airline"]== "BA"]
nodes= [row["Source airport"] for item, row in british_routes.iterrows()]
edges= [(row["Source airport"], row["Destination airport"]) for item, row in british_routes.iterrows()]
british_routes= nx.Graph()
british_routes.add_nodes_from(nodes)
british_routes.add_edges_from(edges)
```

```
plt.rcParams['figure.figsize'] = (16.0, 12.0)
pos = nx.spring_layout(british_routes, iterations=200,
                       seed = 42)
nx.draw(british_routes, pos,
        with_labels=True)
plt.show()
```

Network Representation

Finding Busiest Airports Visually

Suppose our objective of this study is to find which airports are the busiest in a network of airports served by British Airways.



However, we can see that this network is extremely hard to visually interpret. Let us start with doing some experiments with the visualisation parameters.

Network Representation

Finding Busiest Airports Visually

Since, we can see that this network is extremely hard to visually interpret. Let us start with doing some experiments with the visualisation parameters.

```
import matplotlib.pyplot as plt
import seaborn as sns

british_routes= routes[routes["Airline"]== "BA"]
nodes= [row["Source airport"] for item, row in british_routes.iterrows()]
edges= [(row["Source airport"], row["Destination airport"]) for item, row in british_routes.iterrows()]
british_routes= nx.Graph()
british_routes.add_nodes_from(nodes)
british_routes.add_edges_from(edges)
```

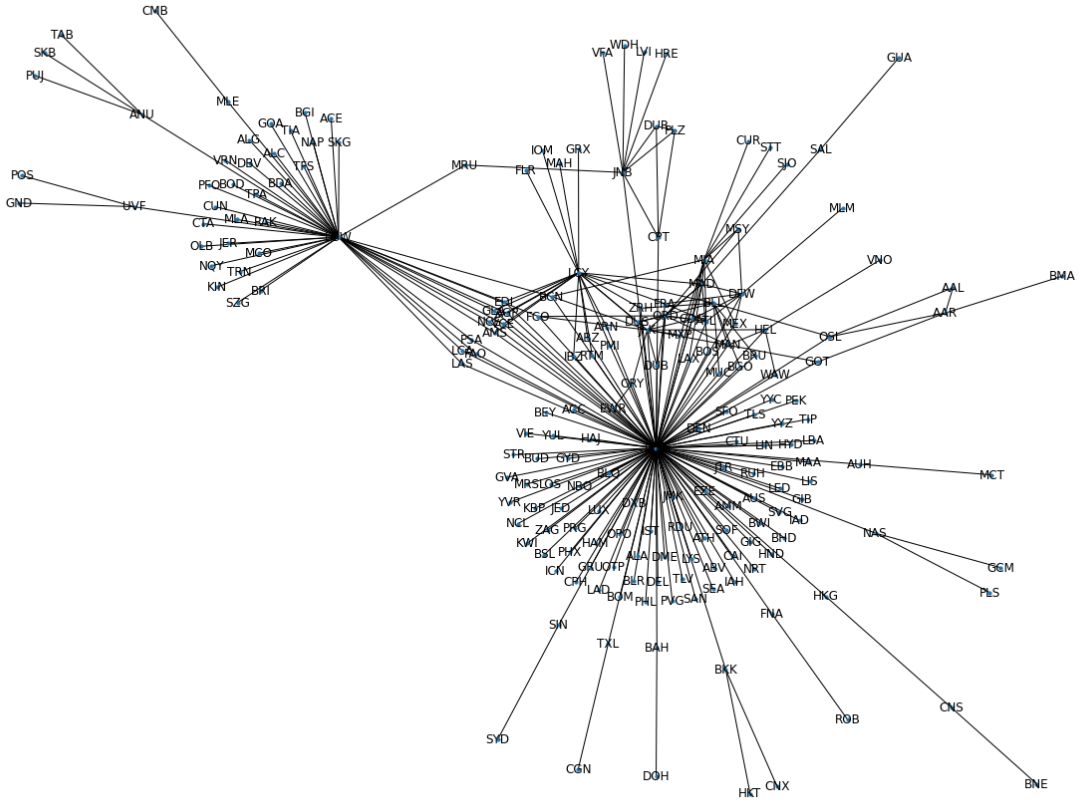
```
plt.rcParams['figure.figsize'] = (16.0, 12.0)
pos = nx.spring_layout(british_routes, iterations=200,
                       seed = 42)
nx.draw(british_routes, pos,
        with_labels=True, node_size= 10)
plt.show()
```

Observe that the node size is the parameter that we have tweaked to improve the visibility. Let us examine how the networks looks like.

Network Representation

Finding Busiest Airports Visually

Now, we can see that this network is a little bit easier to read in terms of the Airport Codes but still hard to visually interpret in terms of the edges. Let us start with doing further experiments with the visualisation parameters.



Network Representation

Finding Busiest Airports Visually

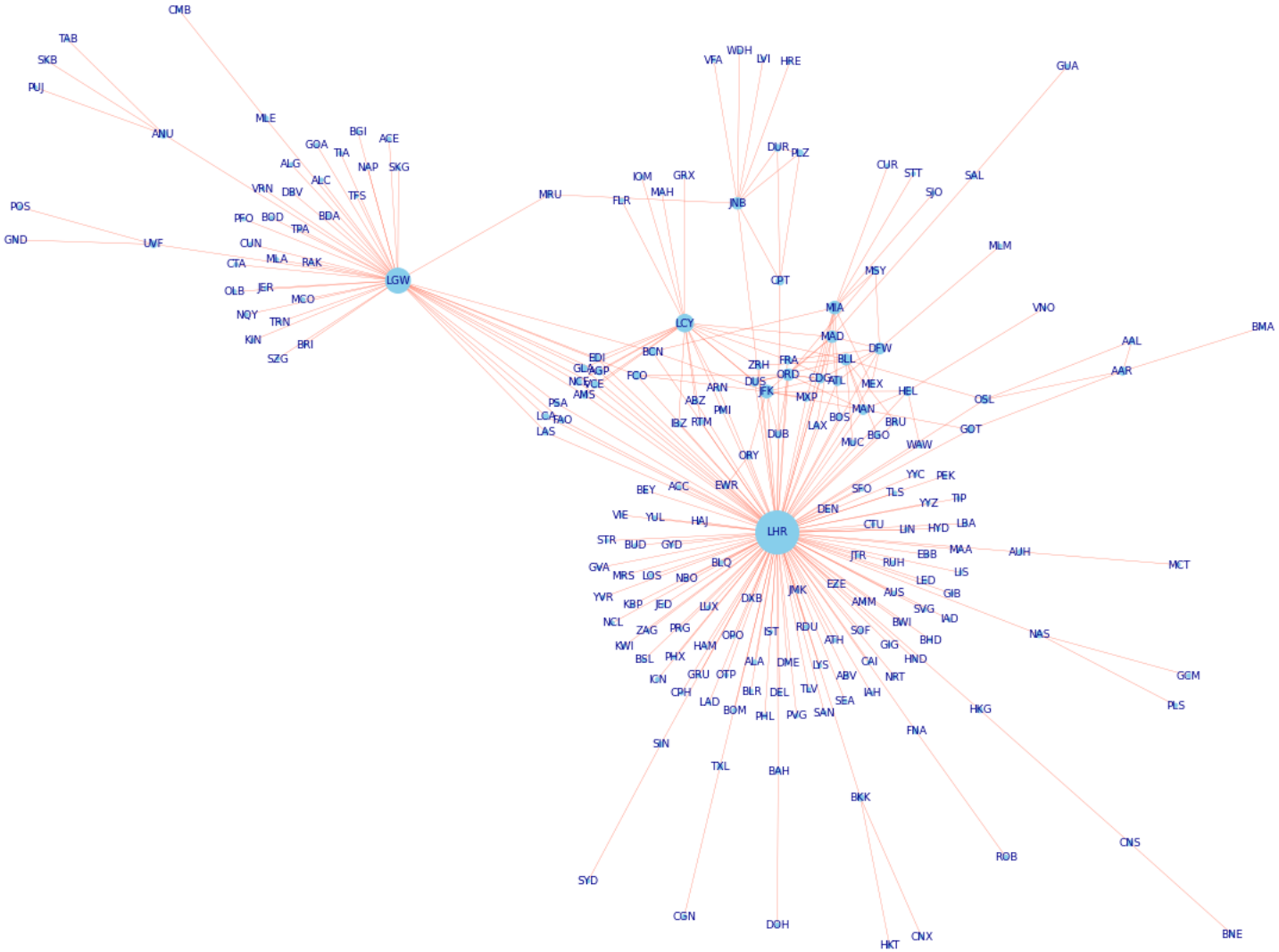
Now, we can see that this network is a little bit easier to read in terms of the Airport Codes but still hard to visually interpret in terms of the edges. Let us start with doing further experiments with the visualisation parameters.

```
import matplotlib.pyplot as plt
import seaborn as sns

british_routes= routes[routes["Airline"]== "BA"]
nodes= [row["Source airport"] for item, row in british_routes.iterrows()]
edges= [(row["Source airport"], row["Destination airport"]) for item, row in british_routes.iterrows()]
british_routes= nx.Graph()
british_routes.add_nodes_from(nodes)
british_routes.add_edges_from(edges)
```

```
plt.rcParams['figure.figsize'] = (16.0, 12.0)
node_sizes= [degree[1]*10 for degree in british_routes.degree]
pos = nx.spring_layout(british_routes, iterations=200,
                       seed = 42)
nx.draw(british_routes, pos, cmap=plt.cm.pink,
        with_labels=True,
        node_color= "skyblue",
        node_size= node_sizes,
        edge_color= "tomato",
        width= 0.3,
        font_size=9,
        font_color= "darkblue")
plt.show()
```

Network Representation

Finding Busiest Airports Visually



Graph Visualization

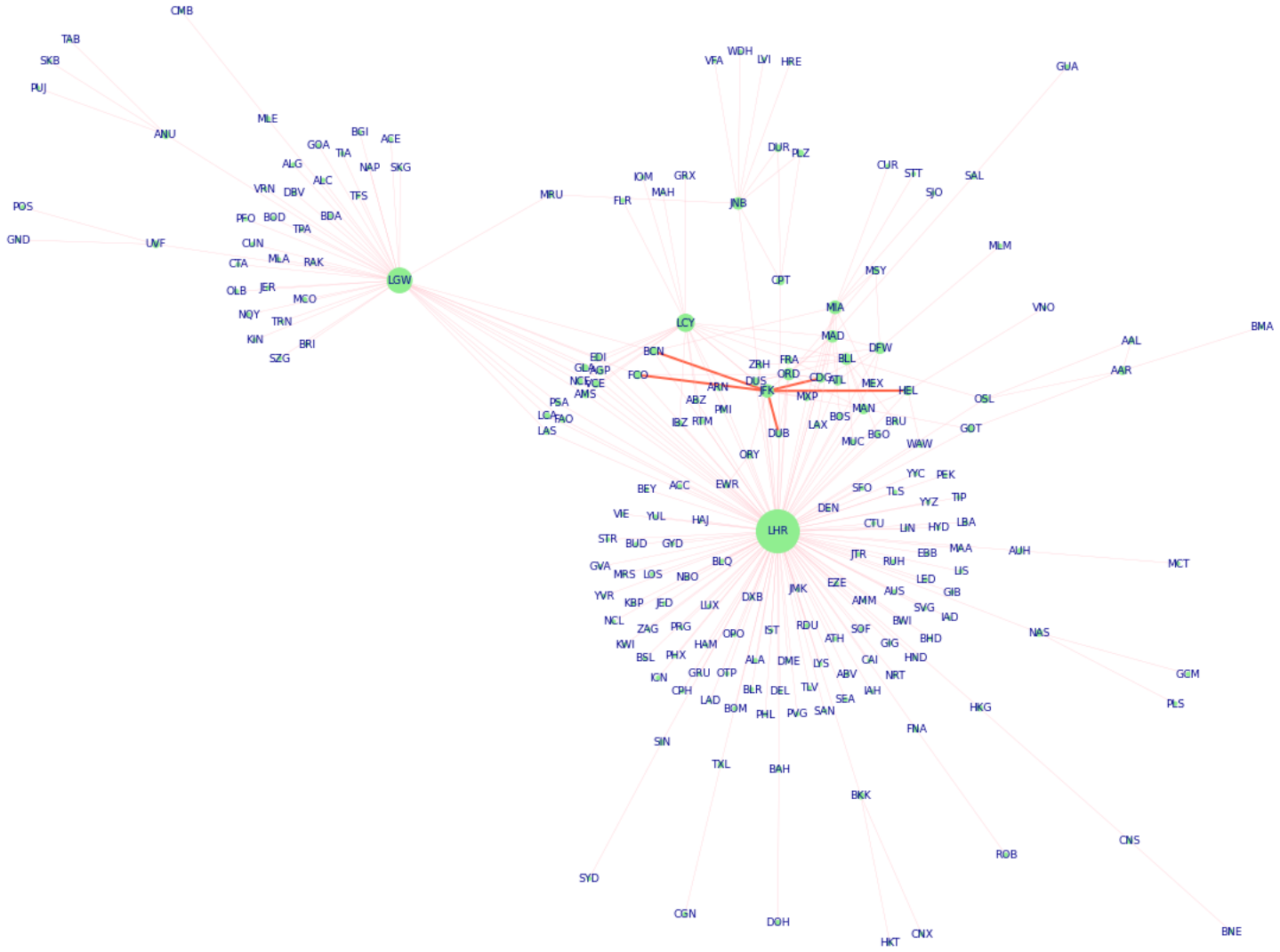
Finding New York Airport Routes Visually

```
import matplotlib.pyplot as plt
import seaborn as sns

british_routes= routes[routes["Airline"]== "BA"]
nodes= [row["Source airport"] for item, row in british_routes.iterrows()]
edges= [(row["Source airport"], row["Destination airport"]) for item, row in british_routes.iterrows()]
british_routes= nx.Graph()
british_routes.add_nodes_from(nodes)
british_routes.add_edges_from(edges)
```

```
plt.rcParams['figure.figsize'] = (16.0, 12.0)
node_sizes= [degree[1]*10 for degree in british_routes.degree]
weights= [2 if edge[1]== "JFK" else 0.25 for edge in british_routes.edges]
edge_color= ["lightpink" if edge[0]== "JFK" else "tomato" for edge in british_routes.edges]
pos = nx.spring_layout(british_routes, iterations=200,
                       seed = 42)
nx.draw(british_routes, pos, cmap=plt.get_cmap('viridis'),
        with_labels=True,
        node_color= "lightgreen",
        node_size= node_sizes,
        edge_color= edge_color,
        width= weights,
        font_size=9,
        font_color= "darkblue")
plt.show()
```

Graph Visualization

Finding New York Airport Routes Visually



BTS | Barcelona Technology School

Graph Traversal

How to travel from one node to another in a graph?

Shortest Routes

Finding Shortest Routes and Lengths

Graph traversal is a common feature of most of the network analysis problems. Basically, the objective of the graph traversal is to determine a path to traverse given information about origin and destination nodes.

Now, while there can be many paths between two nodes of interest, often our focus is towards finding the shortest one.

```
print("Shortest route from London, UK to Genoa, Italy: ",
      nx.shortest_path(british_routes, "LHR", "GOA"))
print("Shortest route length (in terms of number of edges): ",
      nx.shortest_path_length(british_routes, "LHR", "GOA"))
```

```
Shortest route from London, UK to Genoa, Italy:  ['LHR', 'AGP', 'LGW', 'GOA']
Shortest route length (in terms of number of edges):  3
```

Shortest Routes

Finding Shortest Routes and Lengths

Graph traversal is a common feature of most of the network analysis problems. Basically, the objective of the graph traversal is to determine a path to traverse given information about origin and destination nodes.

Now, while there can be many paths between two nodes of interest, often our focus is towards finding the shortest one.

```
print("Shortest route from London, UK to Genoa, Italy: ",
      nx.shortest_path(british_routes, "LHR", "GOA"))
print("Shortest route length (in terms of number of edges): ",
      nx.shortest_path_length(british_routes, "LHR", "GOA"))
```

```
Shortest route from London, UK to Genoa, Italy:  ['LHR', 'AGP', 'LGW', 'GOA']
Shortest route length (in terms of number of edges):  3
```

Even after finding a shortest path, sometimes, it is a question of significance to find other alternative paths of similar complexity. This feature is provided by function to determine all possible shortest simple paths between the two nodes of interest. Now that we already know what is a shortest path, a simple path is a path with no repeated nodes. Let us see how we can do that programmatically,

```
shortest_paths= nx.shortest_simple_paths(british_routes, "LHR", "GOA")
count= 0
for element in shortest_paths:
    if count< 10:
        print(element)
        count+= 1
    else:
        break
```

Here, the object "shortest_path" is a generator object.

```
['LHR', 'AGP', 'LGW', 'GOA']
['LHR', 'AMS', 'LGW', 'GOA']
['LHR', 'BCN', 'LGW', 'GOA']
['LHR', 'EDI', 'LGW', 'GOA']
['LHR', 'FAO', 'LGW', 'GOA']
['LHR', 'FCO', 'LGW', 'GOA']
['LHR', 'GLA', 'LGW', 'GOA']
['LHR', 'LAS', 'LGW', 'GOA']
['LHR', 'LCA', 'LGW', 'GOA']
['LHR', 'NCE', 'LGW', 'GOA']
```

BTS | Barcelona Technology School

Influence Estimation

How to determine relative significance of nodes?

Centrality in Networks

Determining Centrality in Networks: Degree Centrality

It is the simplest to understand measure of centrality based on the concept of links incident upon a node (i.e. the number of ties a node has).

The degree can be interpreted in terms of the immediate risk of a node for catching whatever is flowing through the network (such as a virus, or some information).

In the case of a directed network (where ties have direction), we usually define two separate measures of degree centrality, namely indegree and outdegree.

Accordingly, indegree is a count of the number of ties directed to the node and outdegree is the number of ties that the node directs to others.

When ties are associated to some positive aspects such as friendship or collaboration, indegree is often interpreted as a form of popularity, and outdegree as gregariousness.

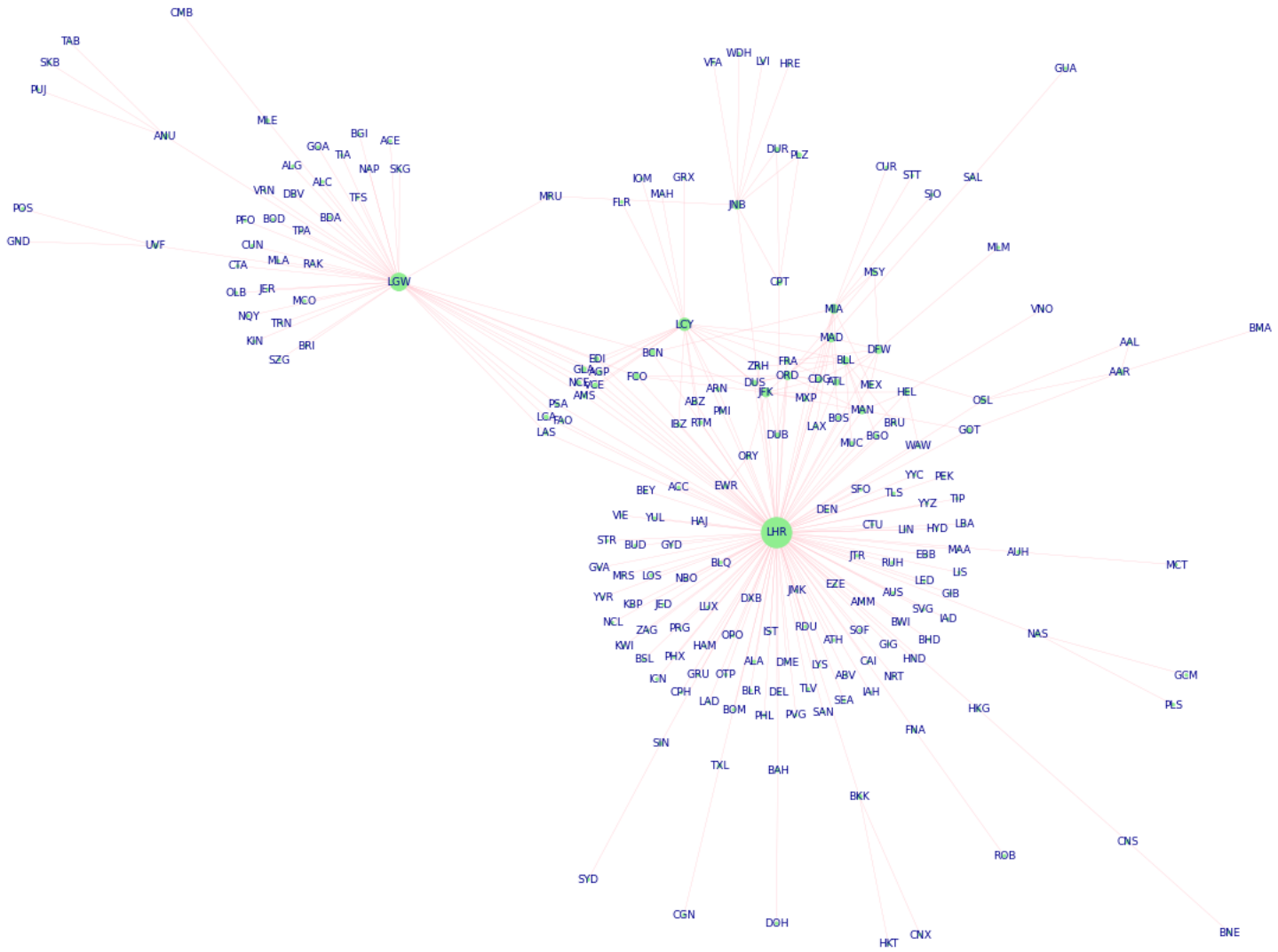
Centrality in Networks

Determining Centrality in Networks: Degree Centrality

```
plt.rcParams['figure.figsize'] = (16.0, 12.0)
degree_centrality= nx.degree_centrality(british_routes)
node_sizes= list(degree_centrality.values())
node_sizes= [val*1000 for val in node_sizes]
pos = nx.spring_layout(british_routes, iterations=200,
                       seed = 42)
nx.draw(british_routes, pos, cmap=plt.get_cmap('viridis'),
        with_labels=True,
        node_color= "lightgreen",
        node_size= node_sizes,
        edge_color= "lightpink",
        width= 0.25,
        font_size=9,
        font_color= "darkblue")
plt.show()
```

Centrality in Networks

Determining Centrality in Networks: Degree Centrality



Centrality in Networks

Determining Centrality in Networks: Betweenness Centrality

It is a measure of centrality based on the concept of shortest paths.

For every pair of vertices in a connected graph, there exists at least one shortest path between the vertices such that either the number of edges that the path passes through (for unweighted graphs) or the sum of the weights of the edges (for weighted graphs) is minimized.

The betweenness centrality for each vertex is the number of these shortest paths that pass through the vertex.

From a macroscopic perspective, bridging positions or "structural holes" (indicated by high betweenness centrality) reflect power, because they allow the person on the bridging position to exercise control (e.g., decide whether to share information or not) over the persons it connects between.

For example, in a telecommunication network, a node with higher betweenness centrality would have more control over the network, because more information will pass through that node.

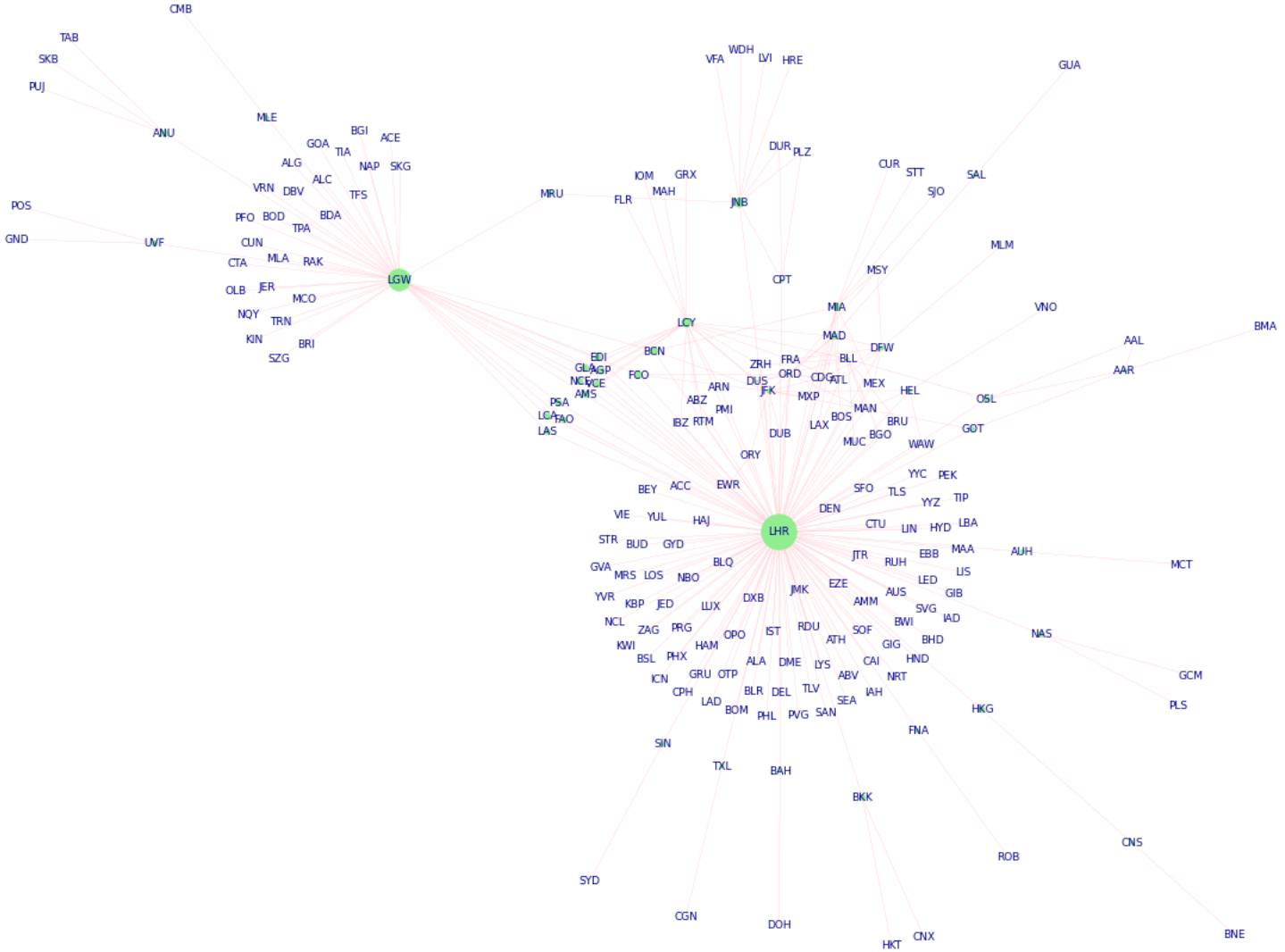
Centrality in Networks

Determining Centrality in Networks: Betweenness Centrality

```
betweenness_centrality= nx.betweenness_centrality(british_routes, seed= 42)

plt.rcParams['figure.figsize'] = (16.0, 12.0)
node_sizes= list(betweenness_centrality.values())
node_sizes= [val* 1000 for val in node_sizes]
pos = nx.spring_layout(british_routes, iterations=200,
                       seed = 42)
nx.draw(british_routes, pos, cmap=plt.get_cmap('viridis'),
        with_labels=True,
        node_color= "lightgreen",
        node_size= node_sizes,
        edge_color= "lightpink",
        width= 0.25,
        font_size=9,
        font_color= "darkblue")
plt.show()
```

Centrality in Networks

Most Important Airports: Betweenness Centrality



Centrality in Networks

Determining Centrality in Networks: Closeness Centrality

It is a measure of centrality based on the concept of shortest paths and defined as the reciprocal of the sum of the length of the shortest path between the node and all other nodes in the graph.

Thus, the more central a node is, the closer it is to all other nodes.

Now, if we consider the network of some infrastructure, say a complex sub-urban metro system then the more central a station is, the easier it will be to reach to other stations from there.

In other words, if the station is falling centrally, then most of the other stations may be reached from there in a degree of a one or two as compared to four or five in case of a non-central station.

Therefore, if we consider the formula, a more centrally located station will be having a smaller sum of length of shortest path between such a station and all other stations resulting in a higher value of the reciprocal.

.

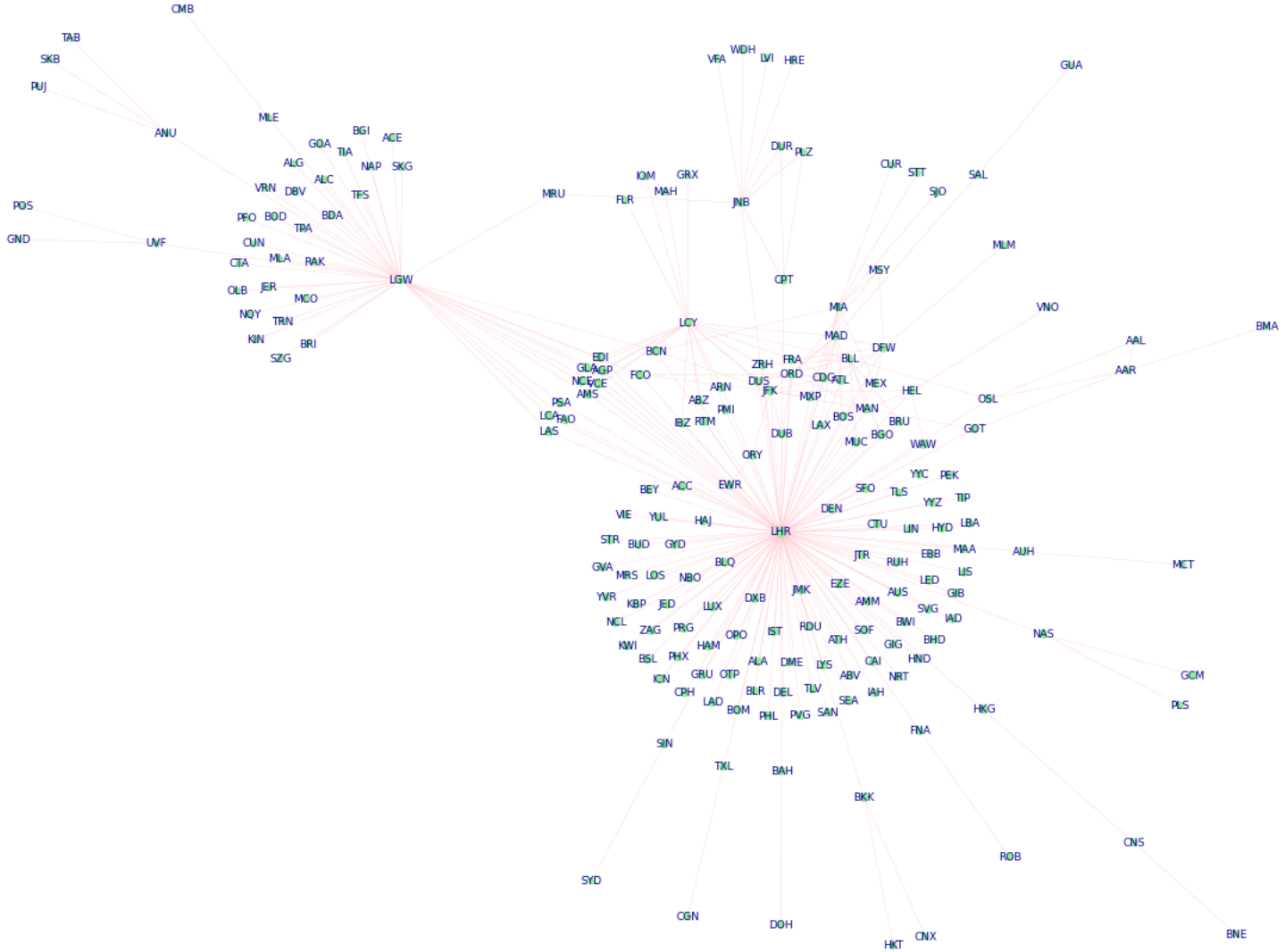
Centrality in Networks

Determining Centrality in Networks: Closeness Centrality

```
plt.rcParams['figure.figsize'] = (16.0, 12.0)
closeness_centrality= nx.closeness_centrality(british_routes)
node_sizes= list(closeness_centrality.values())
node_sizes= [val*100 for val in node_sizes]
pos = nx.spring_layout(british_routes, iterations=200,
                       seed = 42)
nx.draw(british_routes, pos, cmap=plt.get_cmap('viridis'),
        with_labels=True,
        node_color= "lightgreen",
        node_size= node_sizes,
        edge_color= "lightpink",
        width= 0.25,
        font_size=9,
        font_color= "darkblue")
plt.show()
```

Centrality in Networks

Most Important Airports: Closeness Centrality



Centrality in Networks

Determining Centrality in Networks: Eigenvector Centrality

It is a measure of centrality based on the concept of eigenvectors.

Relative scores are assigned to all nodes in the network based on the concept that connections to high-scoring nodes contribute more to the score of the node in question than equal connections to low-scoring nodes.

A high eigenvector score means that a node is connected to many nodes who themselves have high scores.

Google's PageRank algorithm is an example this kind of centrality measure.

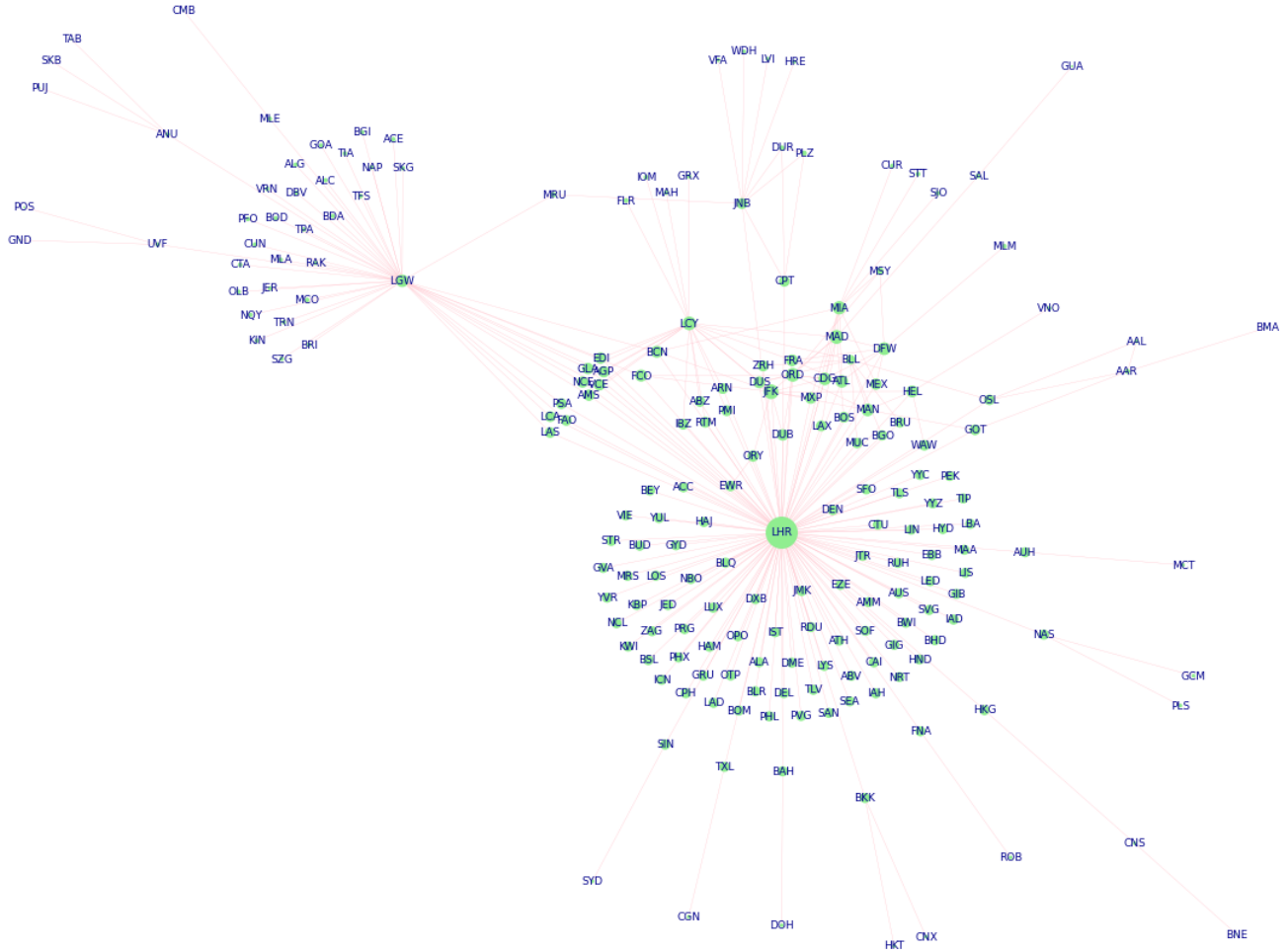
Centrality in Networks

Determining Centrality in Networks: Eigenvector Centrality

```
plt.rcParams['figure.figsize'] = (16.0, 12.0)
eigenvector_centrality= nx.eigenvector_centrality(british_routes)
node_sizes= list(eigenvector_centrality.values())
node_sizes= [val*1000 for val in node_sizes]
pos = nx.spring_layout(british_routes, iterations=200,
                       seed = 42)
nx.draw(british_routes, pos, cmap=plt.get_cmap('viridis'),
        with_labels=True,
        node_color= "lightgreen",
        node_size= node_sizes,
        edge_color= "lightpink",
        width= 0.25,
        font_size=9,
        font_color= "darkblue")
plt.show()
```

Centrality in Networks

Most Important Airports: Eigenvector Centrality



BTS | Barcelona Technology School

Graph Clustering

Clustering in Networks

Detecting Communities

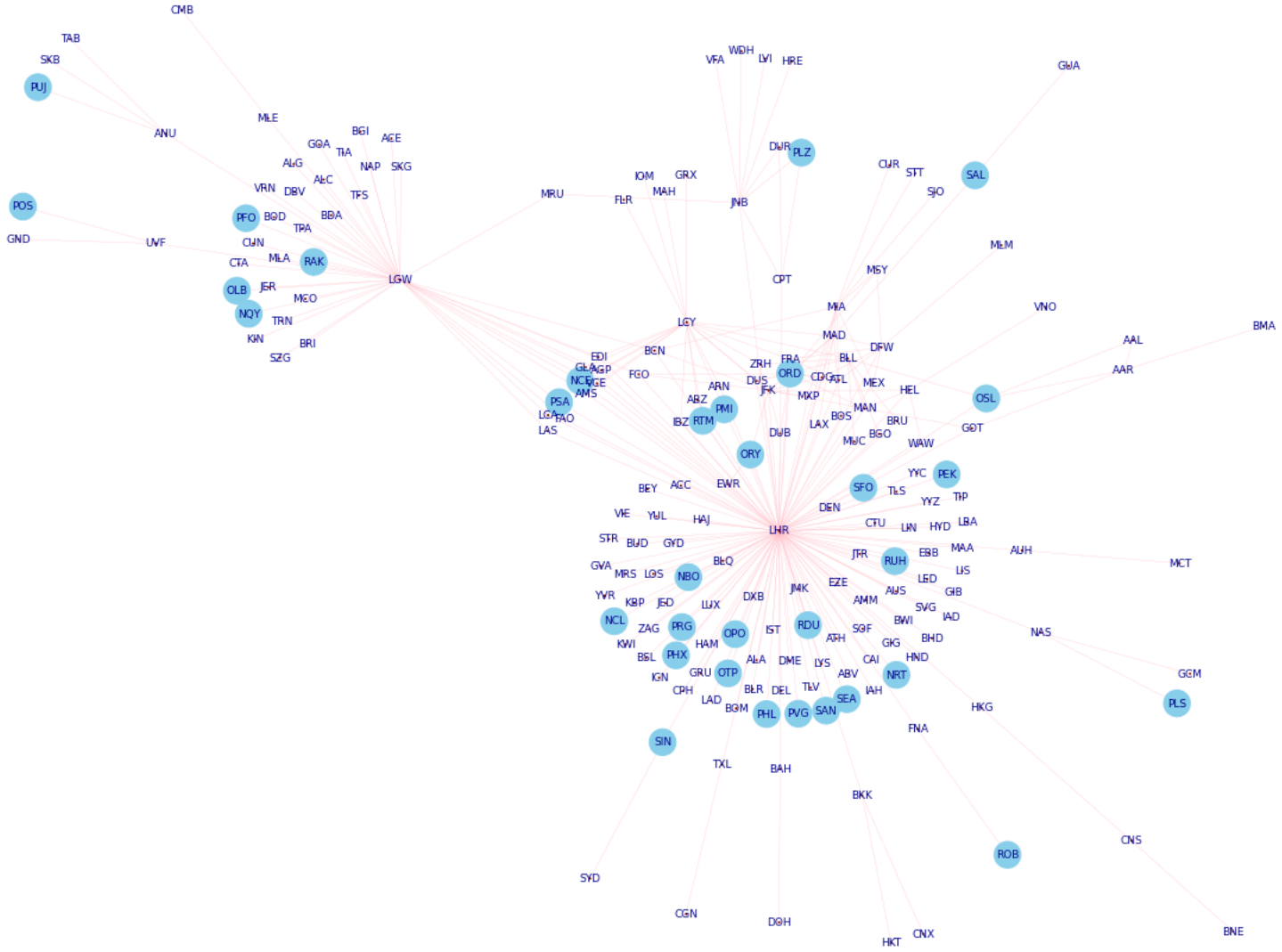
Suppose, our task is to assign different airports into a finite set of K groups or clusters. In such cases, Community Detection is a great feature provided by the NetworkX. Let us see the feature in action.

```
node_colors= ["skyblue" if "GOA" in community else "maroon"
 for community in list(nx.community.asyn_lpa_communities(british_routes, seed= 42)) for node in community]
node_sizes= [500 if "GOA" in community else 1
 for community in list(nx.community.asyn_lpa_communities(british_routes, seed= 42)) for node in community]
```

```
plt.rcParams['figure.figsize'] = (16.0, 12.0)
pos = nx.spring_layout(british_routes, iterations=200,
                       seed = 42)
nx.draw(british_routes, pos, cmap=plt.get_cmap('viridis'),
        with_labels=True,
        node_color= node_colors,
        node_size= node_sizes,
        edge_color= "lightpink",
        width= 0.25,
        font_size=9,
        font_color= "darkblue")
plt.show()
```

Clustering in Networks

Detecting Communities



BTS | Barcelona Technology School

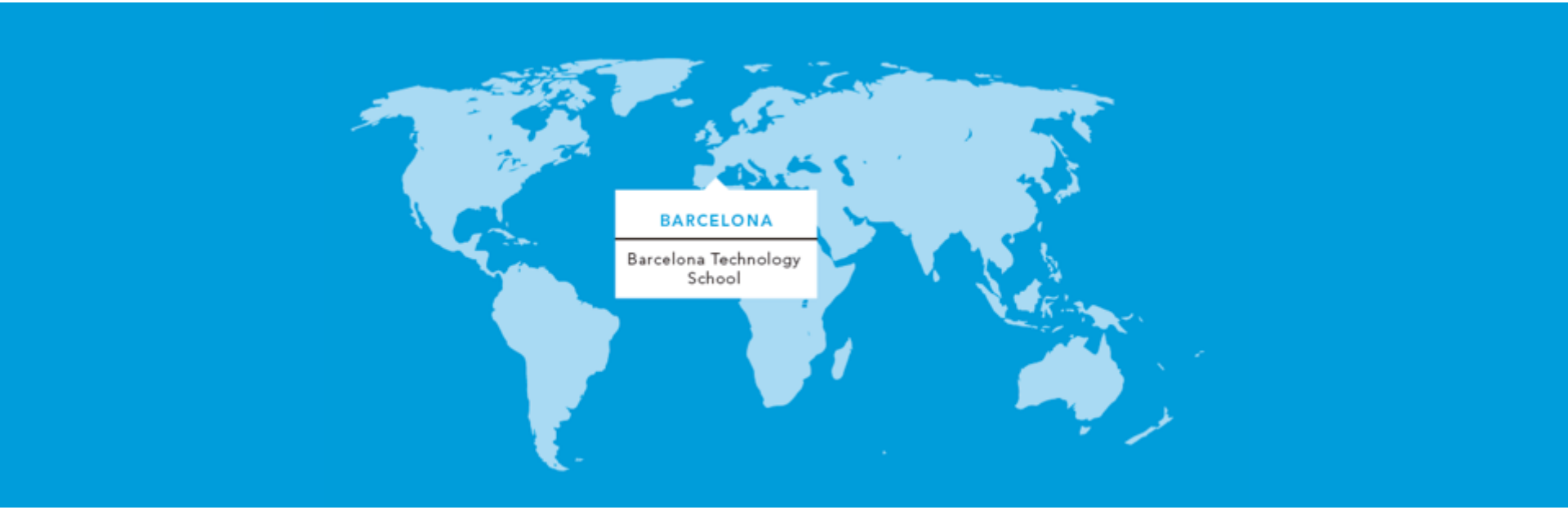
References

References

Additional resources and material

For a more in-depth study of the subject, the following courses and their lecture material as well as references maybe of useful-

1. Networks, Crowds and Markets: Reasoning About a Highly Connected World by David Easley and Jon Kleinberg from Cornell University
2. Complex and Social Networks by Ramon Ferrer-i-Cancho and Argimiro Arratia from Universität Politecnica de Catalunya, Barcelona, Spain
3. Network Analysis and Modeling by Aaron Clauset from Santa Fe Institute, United States
4. Machine Learning with Graphs by Jure Leskovec from Stanford University
5. Graphs and Networks by Dan Spielman from Yale University
6. Spectral Graph Theory by Dan Spielman from Yale University


BARCELONA
Barcelona Technology School