SCHOOL OF PHYSICS AND ASTRONOMY

YEAR 3 PROJECT REPORT

SESSION 2019-2020

Name:	**Kavian Shirkoohi**
Student Number:	**1770404**
Degree Programme:	**BSc Theoretical and Computational Physics**
Project Title:	**Investigations of Complex Networks**
Supervisor:	**Prof J E Macdonald**
Assessor	

Declaration:

I have read and understand Appendix 2 in the Student Handbook: "Some advice on the avoidance of plagiarism". I hereby declare that the attached report is exclusively my own work, that no part of the work has previously been submitted for assessment (although do note that material in "Interim Report" may be re-used in the final "Project Report" as it is considered part of the same assessment), and that I have not knowingly allowed it to be copied by another person.

Investigations of Complex Networks

School of Physics and Astronomy, Cardiff University
May 2020
C1770404

PX3315 Project Report
Word count: 7422 words

Abstract

Developing software to investigate complex networks and their non-trivial properties such as degree distribution, robustness, assortativity and clustering factor. Brief exploration into the application of complex networks in real life phenomena and an analysis of the application.

Table of Contents

1 Introduction

What is a network?

A **network** is a mathematical model describing a system of data points connected by some relation. Networks are useful for representing large data sets, where connections are less obvious and features cannot be reasonably deduced by inspection.

Data points are referred to as **nodes**, and the connections between them **arcs**, with the overall network known as a **graph**. The same data can be represented in a variation of different topologies, known as **trees**. There can be more than one tree per graph.

In the simplest graphs, arcs contain distinct information. It is said that the arcs are **weighted**. Weights assign a value to arcs and can represent real life quantities such as the distance between two nodes, the fuel costs between two destinations on a map, or the time taken to travel two points. Arc weights do not necessarily have to be drawn to scale.

To reach one node from another, the graph can be traversed through a selection of connected arcs known as a **path**. Arcs can be directed, in this case the graph becomes a **digraph** and traversing directed arcs is one-way.

It is no coincidence that the concept of a network and its properties are not unlike that of a real-world network: such is the case of a transportation network, with nodes representing intersections and arcs the roads between them. Each intersection can have a number connected roads. The amount of connections to other nodes at any given node is known as the **degree** (or **order**) of the node. Many transportation networks were first modelled as a network before being implemented into urban planning. A famous example is the London Underground train system[1].

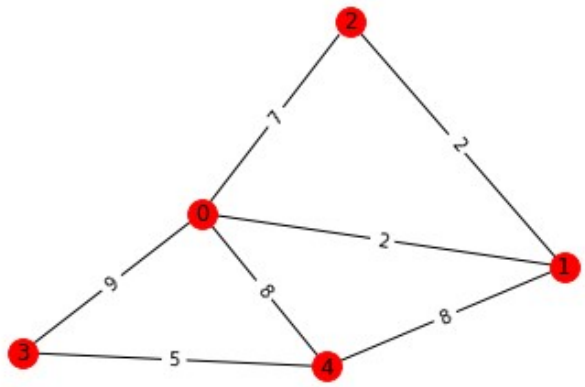


Figure 1.1. A simple graph with N = 5 nodes. There will always be a minimum of N + 1 arcs on the network, and a maximum degree of N - 1 per node. Traversing is simple: to arrive at node 3 from node 1 using the shortest path, one would traverse arcs 1-0 and 0-3. The total length of the shortest path would be the total arc weights traversed, 2 + 9 = 11. Source: NetworkX.

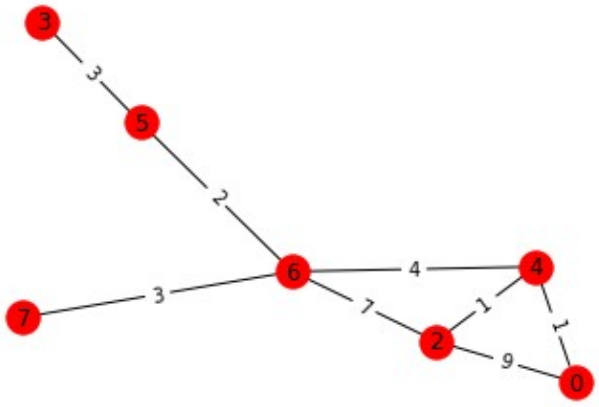
The two main forms of simple networks are **random networks** and **lattices**. The former consists of data points randomly connected to each other according to a network-wide probability of **connectivity** between the nodes, *P*. The latter involves a repeating periodic structure, that is commonly found in Bravais lattices[2]. In this study, only random networks will be considered.

Representing a network in a computer program

A network can be represented as discrete data by an **adjacency matrix**. The rows and columns represent nodes, with non-zero elements representing arcs and their weights. There can be more than one arc between the same two nodes, however this case will not be explored in this study.

The Python 3.0 program *random_network.py* (see Appendix A.1) contains an implementation of an adjacency matrix to represent a simple random network. The network is initialised with *N* nodes and a connection probability *P*.

In this study, the following libraries will be utilised to aid representation and manipulation of networks: Scipy, Numpy, Matplotlib and NetworkX. The former contain essential mathematical functions and the latter have powerful procedures for drawing networks as graphs and quickly analysing them.



	0	1	2	3	4	5	6	7
0	7	0	0	0	1	0	0	0
1	0	0	0	0	0	0	0	0
2	9	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	1	0	0	0	4	0
5	0	0	0	3	0	0	0	0
6	0	0	7	0	0	2	0	0
7	0	0	0	0	0	0	3	0

Figure 1.2. A network generated with its corresponding adjacency matrix. Here, N = 7 and P = 0.2. Each row and column on the matrix represent a node, with the intersection element forming a connection between the two nodes. Source: NetworkX.

2 Complex networks

What makes a network complex?

In essence, a complex network is one with non-trivial topological properties that do not occur in simple random or lattice networks. The properties of a complex network become interesting when used to represent real-life data sets, presenting useful correlations between the relational data and network behaviour. It is this particular area of interest that this study will focus on mostly. When handling large data sets, the properties of a network can no longer be ascertained by inspection and a methodical approach is preferred.

There are two primary classes of complex networks, scale free networks and small world networks. Both feature key properties that make them complex, such as having a **power-law** degree distribution, a high **clustering coefficient** (tendency for nodes to cluster together to form 'cliques'), **assortativity** or **dissasortativity** between vertices (nodes of similar degree are more likely to link to similar degree nodes), and **hierarchical structure** (any arrangement of nodes with a higher or lower order of accompanying nodes in the network).

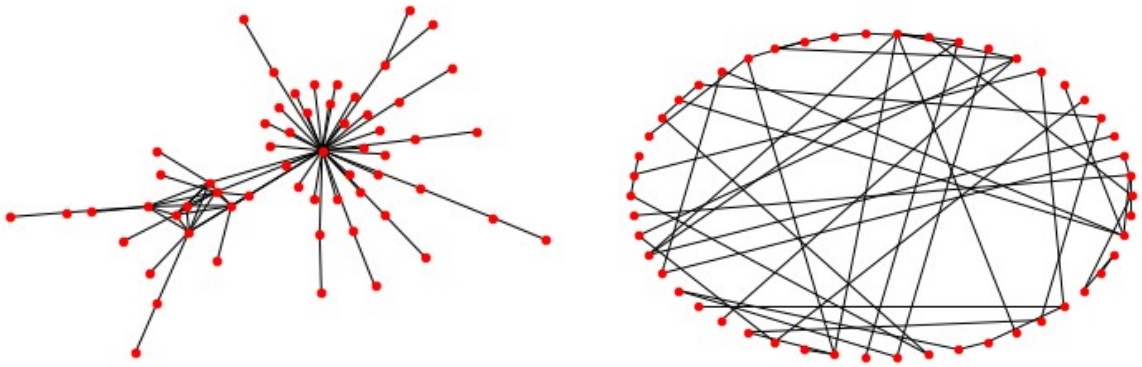


Figure 2.1. Two examples of complex network, the scale free network (left) and the small world network (right). When the network grows considerably large, it makes sense to approach analysis of the network in a computational manner. Features that seem obvious now may be more difficult to identify as the network grows. Source: NetworkX.

Complex networks are commonly studied for their **robustness**, that is, how adverse the consequences of an 'attack' are (a node or edge is removal from a network). The more robust a network, the less of an effect an attack has. Networks can survive different types of attacks, from **random** attacks (removal of a node or edge at random) to **targeted** attacks (removal of a high ranking node i.e. one with high degree) and the resultant network after an attack is frequently analysed to identify links that are disconnected. An example of network robustness is the evaluation of power grid resilience[3]; how vulnerable a power grid is to a failure of one of the pylons or transformers (with some elements ranking higher in importance than the rest).

Scale free networks

A scale free network is any network that, upon addition of new nodes, has a preference to connect the new nodes to existing nodes of high degree. The degree distribution follows a power-law with network growth and has no characteristic scale. This implies that a few nodes have a much higher degree than the rest on the network. Scale free networks are characterised by their tendency to contain 'hubs' of nodes (nodes which hold more connections towards the same high-degree node than the rest of the network).

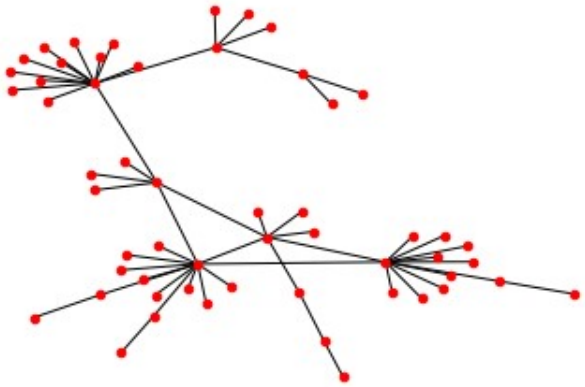


Figure 2.2. A typical scale-free network characterised by a few high degree hubs forming clusters connecting the majority of the lower degree nodes. Source: NetworkX.

The property of assortativity is obvious from the cluster-like nature of the graph. The fraction of nodes *P(k)* within the whole network having *k* connections, for large values of *k*, is expressed mathematically by (1)

$$P(k) = k^{-\gamma} \tag{1}$$

Here, γ is any parameter such that $2 < \gamma < 3$. This relationship outlines the evolution of the network with respect to adding more nodes demonstrating how degree distribution increases exponentially[4]. A graphical analysis of this relationship will be explored later in this study.

Scale free networks have many applications. Studies involving scale-free networks have included social networks in high school, the number of paper citations per academic (following a power-law according to the Price[5] model), a primitive model of the world wide web[6], and the number of sexual partners between an individual (if a given individual previously had many partners, it is more likely they will sleep with more people than those who have not).

Small world networks

A small world network is a network that is subject to the small world effect. The small world effect proposes that two nodes, at random, within a large network of nodes and arcs are connected by less than six other arcs[7]. This is known as the six degrees of separation theory. The dynamics of the small world effect require a very high **connectivity** between nodes within the network. When optimising in this study, the emphasis is on the reduction of traversal steps to reach one point on the graph from another, rather than the weight of the total path.

Consider a ring network of nodes, i.e. nodes connected to their nearest neighbours, with a low degree distribution. Let *N* be the number of nodes on the network, *K* the number of neighbours each node is connected to and *P* the probability of rewiring each arc in the network.

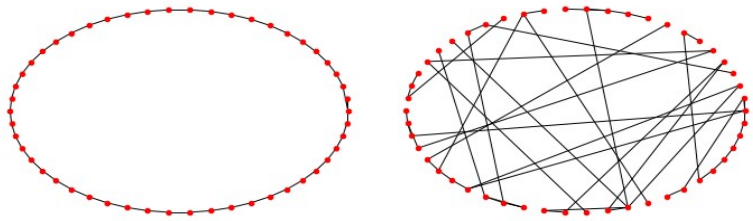


Figure 2.3. A ring network, before (left) and after (right) the small world algorithm is applied. In a ring network each node is connected to at most, its two adjacent neighbours. The small world algorithm re-wires neighbour connections randomly. Source: NetworkX.

In Figure 2.3, it is apparent that several arcs would need to be traversed to move to different nodes in the network. In most cases, a section of whole ring would need to be traversed. The small world network re-wires adjacent connections between nodes to other nodes at random, creating alternative paths of traversal. Within the large limit for iteration of the algorithm, this effect eventually reduces to the six degrees of separation theory. If the number of *K* neighbours each node is connected to increases, there will be more alternative paths across the network for the same starting point.

Mathematically, the small world network can be defined such that the distance *L* between two randomly chosen nodes grows proportionally to the logarithm of nodes *N* in the network as shown by the relationship (2)

$$L \propto log\ \mathrm{N} \tag{2}$$

The small world network can prove useful where the connection between data points is not immediately obvious, and the number of arcs that will be traversed in a path must be minimised. By re-wiring nodes on the network, connections can provide shortcuts from one area of the graph to another. This yields interesting results for degree distribution, too, which will be investigated later.

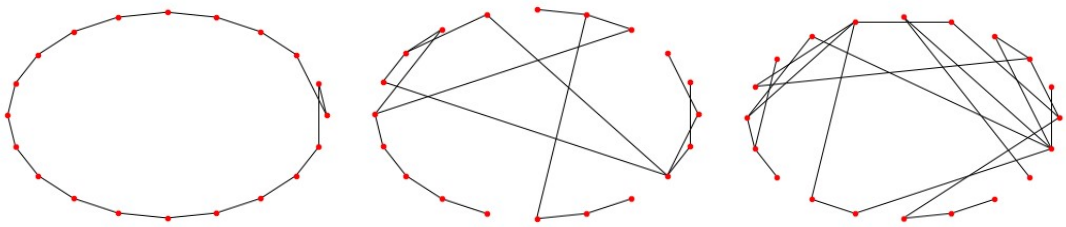


Figure 2.4. As the number of arcs rewired increases, the graph converges towards a random network. Network with no arcs rewired (left), half rewired (centre) and almost all arcs rewired (right). Source: NetworkX.

3 Investigating properties

To investigate the properties of complex networks, the scale free and small world networks are represented in Python code. Appropriate use of the mentioned libraries is made to represent applications of the networks later in this study. Effective utilisation of the adjacency matrix is made to adjust networks as required, and NetworkX functions are used to draw and analyse the properties of the networks as a graph format. See Appendix B for the implementation of the two complex network types in P

The code in the file *network_functionality.py* (see Appendix A.2-4) contains the essential functions used to construct and analyse the complex networks.

Implementing the scale free network

The scale free network is generated with the Barabási-Albert model. This model involves the preferential attachment algorithm. Each node is given a probability proportional to its degree. Each time the algorithm runs, a new node is added to an existing node dependent on the probability.

First, a random network is generated. Following this, the random network is input to the function *create_scale_free_network* which takes the evolution number *S* and assortativity constant *A* as parameters (see Appendix B.1). S is an integer outlining how many new nodes will be added to the network; in effect, how many times the preferential attachment algorithm will run. *A* is the probability that new hubs will be formed and takes a value between 0 and 1.

Upon each iteration, the function ports the elements from the old matrix to the new one before adding another column and row to represent a new node. Then, the nodes on the existing network are evaluated for their degree number *k* (the amount of 1's in their column) with a sum over all existing nodes *j*, and a probability p_i for each node *i* is generated in (3)

(3)

$$p_i = \frac{k_i}{\Sigma_j k_j}$$

A normalised random number is generated (with a range from zero to the highest probability in the distribution) and is evaluated against the probability distribution. A list of possible nodes, *high_nodes* (the nodes with a probability within the range of the random number) is generated and an element is chosen at random to give nodes other than the highest degree node a chance at becoming hubs. If this random selection was not implemented, the network would no longer be a complex network but a simple star network similar to the highest degree node in Figure 3.1.

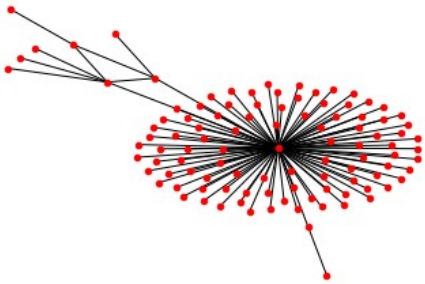


Figure 3.1. A scale-free network with a low assortativity leads to an unlikeliness to form new hubs. If the random selection of possible nodes does not occur, the topology will tend to be similar to one of the hubs pictured. The network would no longer be complex, but a simple star network. Source: NetworkX.

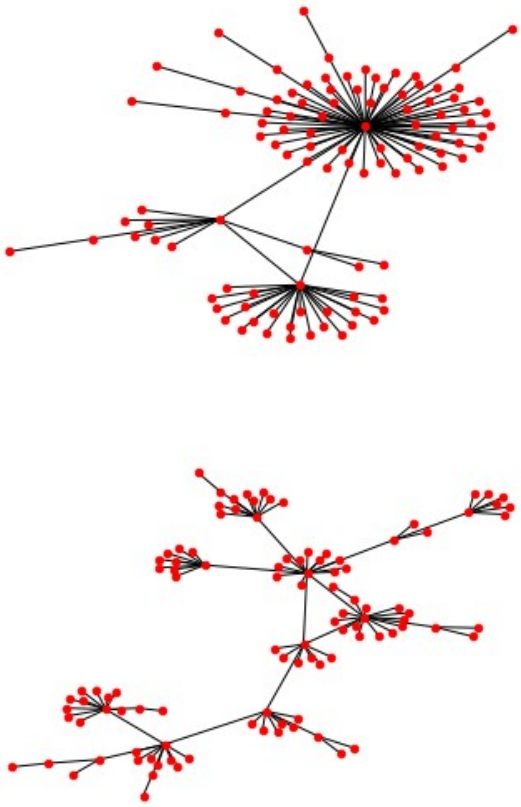


Figure 3.2. Pictured (top) is a network of 100 new nodes, with a an assortativity factor of 0.1. This topology is similar to Figure 3.1 and contains few hubs, creating a high degree-distribution. Contrasted (bottom) to a network with a assortativity factor of 0.9 to create new hubs with a more even degree-distribution. Source: NetworkX.

Implementing the small world network

To begin, a ring network is defined for *N* nodes. These nodes are connected to their adjacent neighbours. As the number of nodes increases, the number of adjacent connections increases but never exceeds three; a node can be connected to its three neighbours at most.

To simulate the small world effect, the Watts-Strogatz model is used in the algorithm (see Appendix B.2). This algorithm is given a rewiring probability parameter, *B*, which is the likeliness of rewiring nodes in the network. The number of arcs chosen to rewire, *chosen_arcs*, is the product of *N* and *B*. The algorithm iterates through the network, rewiring arcs at random. The rewiring process is simply the removal of a connection between a node and its neighbour, and a connecting it to another node chosen at random. The iteration factor *while not (check_node_is_connected(adj_matrix, rewired_node)*) is important to ensure a step has not been skipped just because the chosen node has no neighbour.

The program generates a small world network as expected. It is important to note that when drawing a graph, the third parameter of '1' is passed to *create_graph*. This specifies that the graph to draw is in a circular layout, which is essential for visualising a small world network.

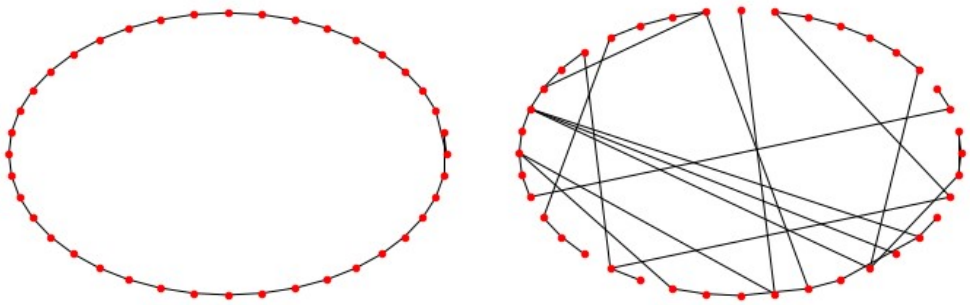


Figure 3.3. N = 40 nodes and B = 0.5. Pictured (left) is the ring network, and (right) the rewired network after scale_free_network has operated on the adjacency matrix. Source: NetworkX.

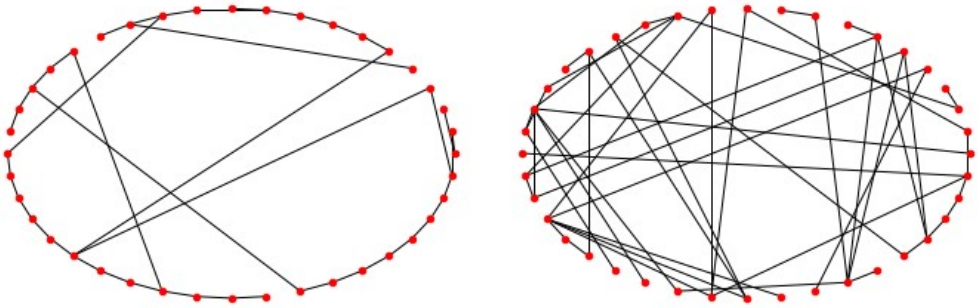


Figure 3.4. The same network with B = 0.2 (left) and B = 0.8 (right). Both extremes of B demonstrate how the network topology evolves with the iteration of arc rewiring. The network clearly becomes more random as more arcs are rewired. Source: NetworkX.

Mean path length

In a small world network, it is interesting to note that the mean path length decreases as more arcs are rewired. This is a good example of demonstrating the 'small world effect'[8]. As the connections between nodes on the network become more random, the number of nodes required to traverse in a path to reach another node decrease.

Mean path length also has other applications. Since it is defined as the average number of steps to reach one node to another, it can be used to optimise information transmission of on a computer network, to reduce power-grid losses[9] or model the efficiency of an organism's molecular mechanisms[10].

The mean path length L is shown to decrease as the probability of rewiring arcs increases. This is demonstrated in Figure 3.5.

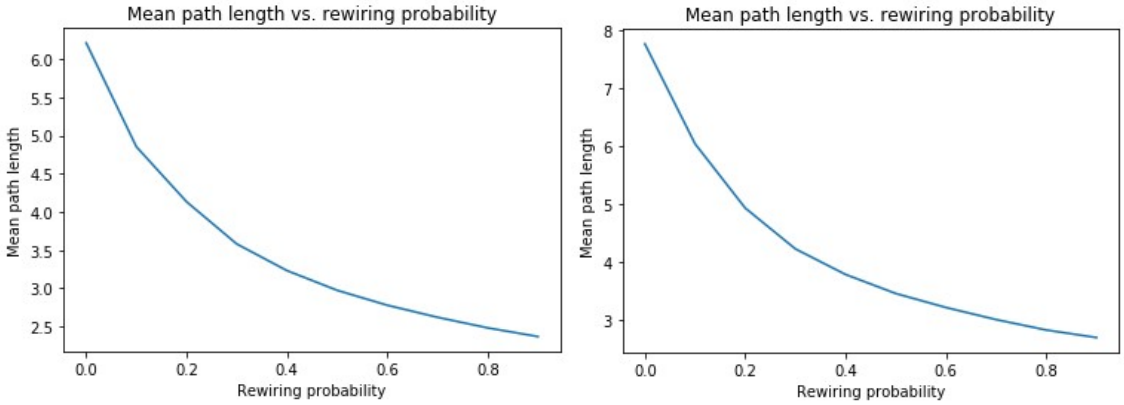


Figure 3.5. Mean path length for a network of 200 nodes (left) and 500 nodes (right) against rewiring probability of 0 to 1. There appears to be an exponential decay relationship between the two. The number of nodes appears to have no effect. Source: Python.

The relationship between mean path length L and the number of nodes N on the network can also be investigated. The results show that (2) is confirmed, as there is a logarithmic relationship between L and N when plotted.

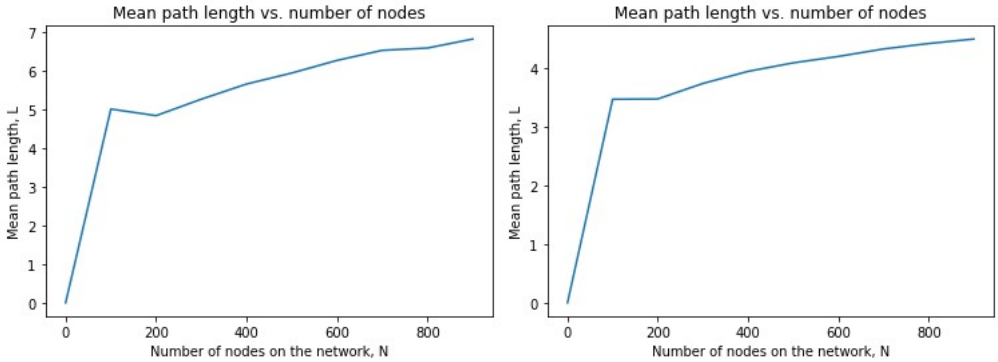


Figure 3.6. Mean path length calculated dependent on N for N = 10 to N = 1000. Pictured (left) is the graph for B = 0.2 and (right) B = 0.8. The sharp tangent between 0-200 on the nodes axis is due nodes having to more adjacent neighbours as N increases. The rewiring probability decreases the average mean path length, likely due to the number of alternative paths available as more arcs are rewired. Source: Python.

For the scale free network, the mean path length can be investigated as the number of nodes added increases. The mean path length increases logarithmically with N. This implies that (2) is also applicable for scale free networks.

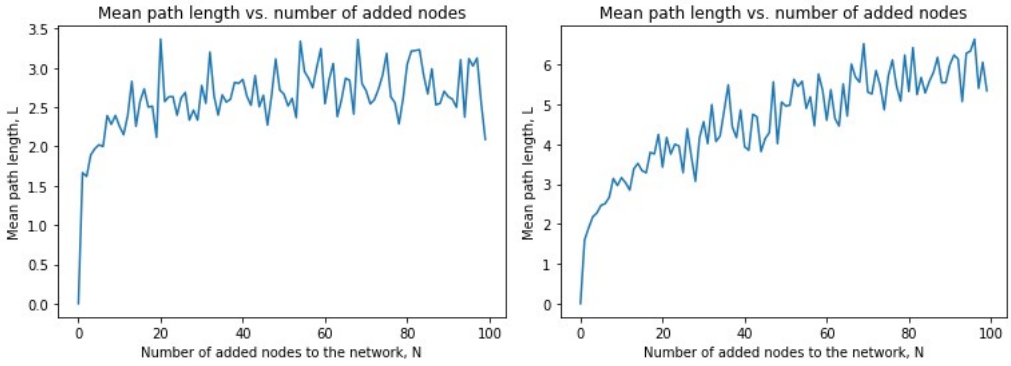


Figure 3.7. Mean path length calculated for A = 0.2 (left) and A = 0.8 (right) from adding 0 to 100 nodes on the network. There is a logarithmic correlation between the two variables for both values of A. The assortativity factor A appears to control the coefficient for the logarithmic gradient – higher values follow the logarithmic profile more closely. Source: Python.

This implies that, for a high assortativity constant *A*, the network becomes similar to that of a ring network.

Degree distribution

One important aspect to investigate for any complex network is its degree distribution. The proportion of nodes with a higher degree over the rest of the network provides a key factor for determining the network's **robustness** – that is, how drastic the effects of removing a specific node or one chosen at random are on a network and its paths – which is an essential factor when modelling a network as real world system[7].

For the scale free network, there is a very high level of degree distribution once the network has evolved over time. A graph of degree distribution with the addition of nodes can be drawn to illustrate the relationship. It helps to visualise the network tree alongside the graph for an indication of what the data represents.

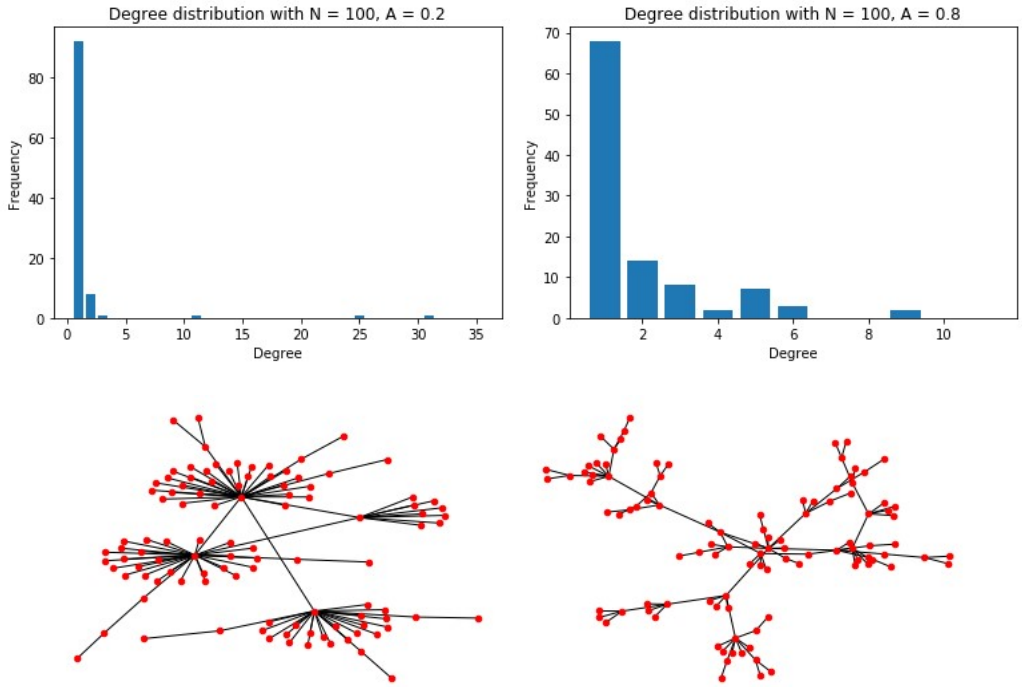


Figure 3.8. Pictured is a network with A = 0.2 (left) and A = 0.8 (right) for 100 nodes. There is a much higher degree distribution in the former, due to the few hubs with many connections to lower-degree nodes. In the latter, there is a lower distribution. This may be due to more nodes sharing a similar degree, since there are more hubs of lower degree. Source: Python.

In the small world network, degree distribution is more even and tends towards a negatively-skewed gaussian as the number of nodes increases.

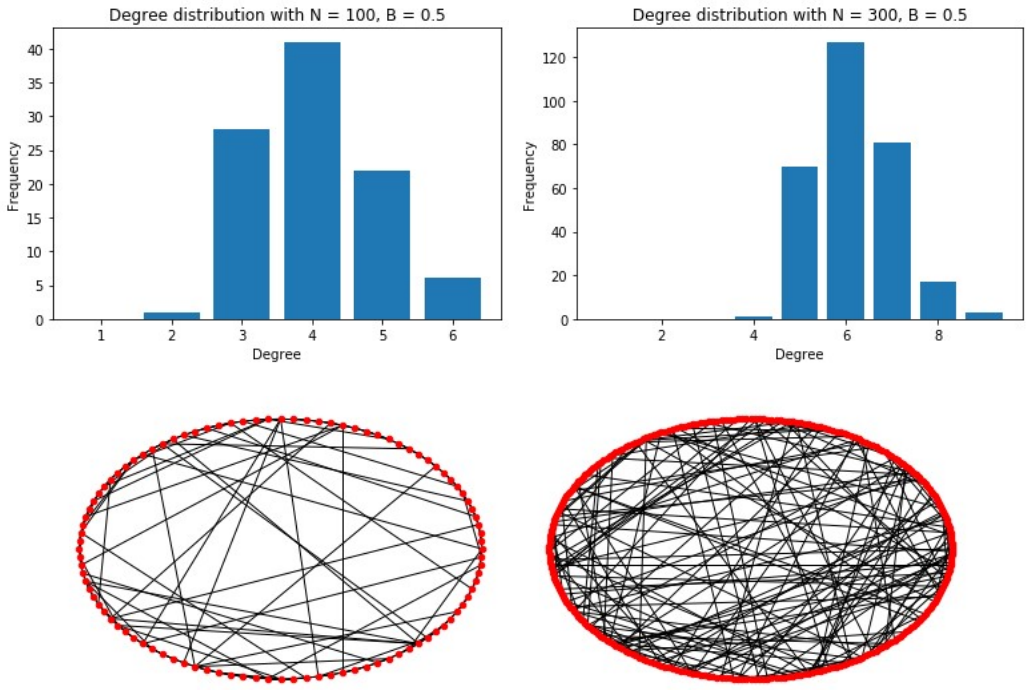


Figure 3.9a. The degree distribution of the scale free network with N = 100 (left) and N = 300 nodes (right), with B = 0.5 fixed. Source: NetworkX.

Whilst keeping N constant at $N = 200$ nodes, the relationship between degree distribution and rewiring probability B can be determined. It follows that the gaussian profile becomes even more pronounced as more arcs are rewired. There is a greater variety in node degree as B increases, since the randomness of the network increases.

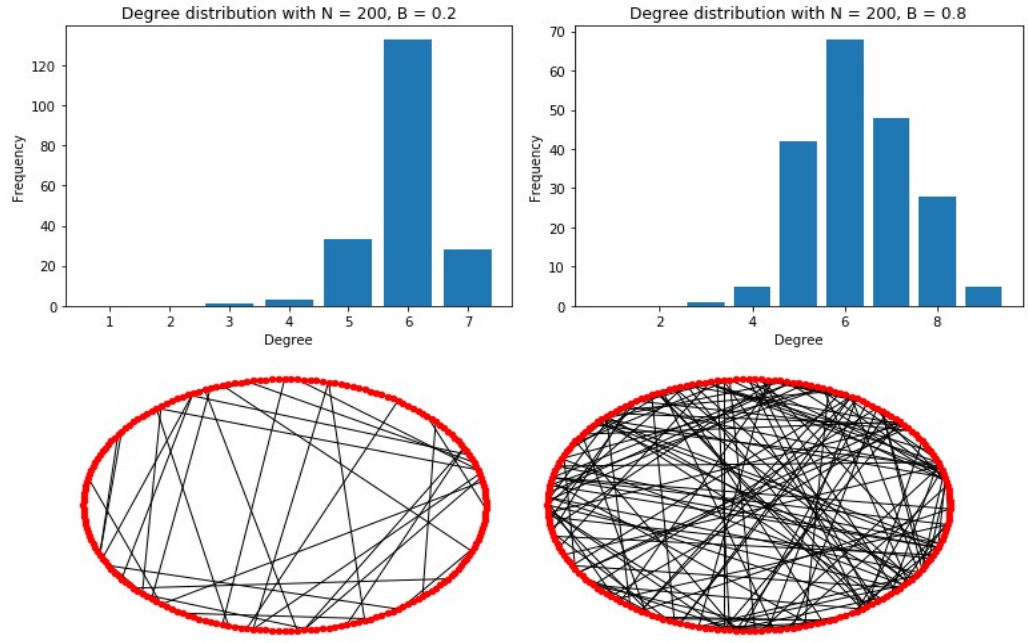


Figure 3.9b. The degree distribution of the scale free network with B = 0.2 (left) and B = 0.8 (right). N fixed at N = 200 nodes. Source: NetworkX.

Measures of centrality

Centrality measures are important for identifying the most important nodes within a network. Measures of centrality originated from social network analysis where key individuals within a network were more influential than others[11]. Although subject to limitations, such as those encountered in the Krackhardt kite graph[12], centrality can provide a good picture of which nodes matter the most.

There are several ways to measure centrality and three of these will be explored: degree, closeness and betweenness centralities. Each of these measures will now be explored for the scale free and small world networks.

Degree centrality. As the name suggests, the degree of a node is determined and can be interpreted in terms of how likely the node may 'catch' whatever is travelling on the network (such as data transmission or a virus). Degree centrality is the most simple measure.
For a node v in a given network G the degree centrality C_D of a node is defined

(3)

$$C_D(v) = \deg(v)$$

Where deg(v) is the degree of the node v. The function *get_degree_centrality* is used to return the degree centrality of a network per node.

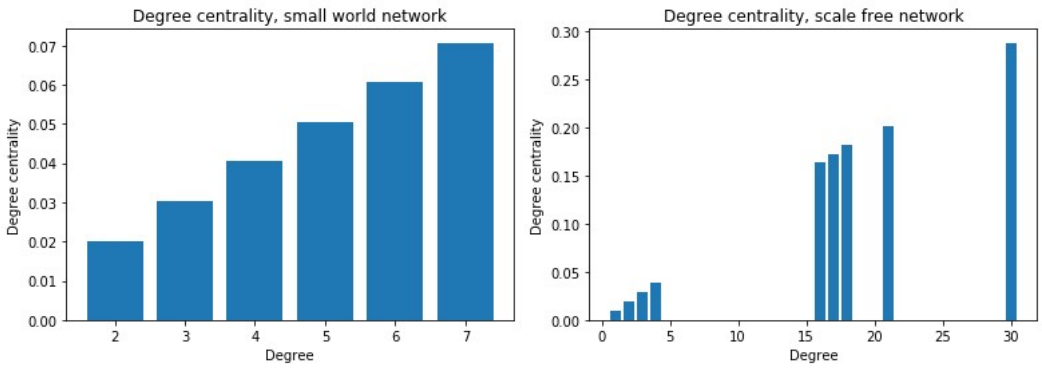


Figure 3.10. For both the small world and scale free networks, there appears to be a positive linear relationship between degree centrality and degree. This impliex that higher degree nodes have a higher centrality ranking. Source: Python.

C**loseness centrality**. It is defined as the average relative length of the shortest path between a node and the other nodes in the network. Closeness $C(x)$ for a node x is defined[13]

(4)

$$C(x) = \frac{1}{\sum_y d(y,x)}$$

Where a sum is made over the distance between the node x and other nodes y. The function *get_closeness_per_node* returns the closeness centrality per node.

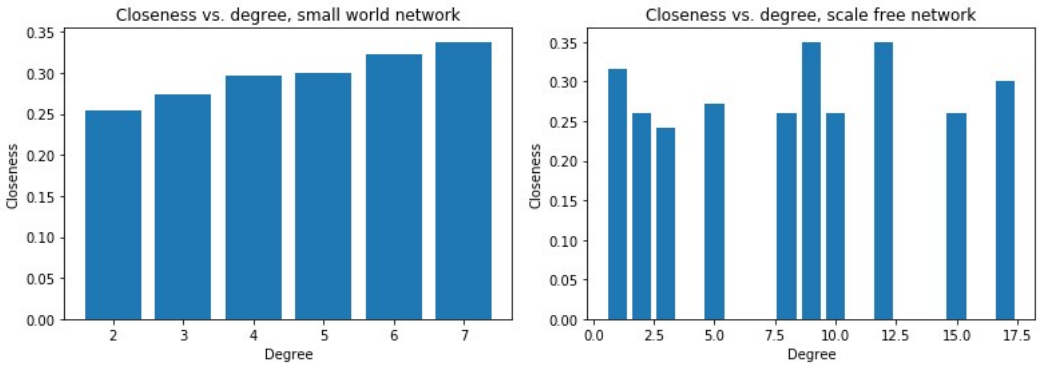


Figure 3.11. Closeness centrality vs. degree appears to have a positive linear relationship in the small world network but becomes more random in the scale free one. This could be due to the wider degree distribution. Closeness therefore is not a good measure of centrality for scale free networks. Source: Python.

Betweenness centrality. It is a measure of how many relative times a given node *v* exists along the shortest path between two other nodes. Betweenness was originally used for expressing the influence a person has on the communication between other people in a social network[14]. The betweenness in a network *G* with *N* nodes can expressed in[15]

(5)

$$C_B(v) = \Sigma \frac{\sigma_{st}(v)}{\sigma_{st}}$$

Where σ_{st} is the total number of shortest paths from *s* to *t* and $\sigma_{st}(v)$ is the number of these paths which include *v*. The summation is over *N* nodes where *s*, *v* and *t* cannot be equal. The function *get_betweenness_per_node* returns betweenness centrality per node.

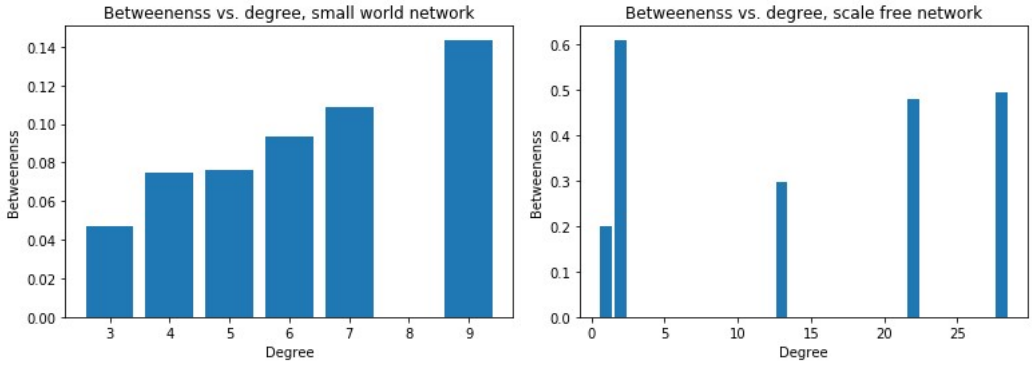


Figure 3.12. Similar to closeness centrality, betwenenss appears to have a positive linear relationship with degre, whereas the scale free network appears to be more random. Possibly due to wider degree distribution in the latter. Source: Python.

Each measure of central tendency has its merits, and it is apparent from the results that these should be selected according to the nature of the network. Degree centrality, due to its simple and universal nature, appears to be the most consistent.

Clustering coefficient

The clustering coefficient is the measure of the degree where nodes cluster together. It implies how likely nodes are to form closely connected 'cliques' between each other and is evaluated per node by considering the number of links between its neighbours. To calculate the cluster coefficient C_v for a node v, the following formula is used

$$C_! = \frac{2 \times N_!}{k_! \ (k_! - 1)} \tag{6}$$

Where k_v is the degree of the neighbour node and N_v is the number of links between the neighbours.

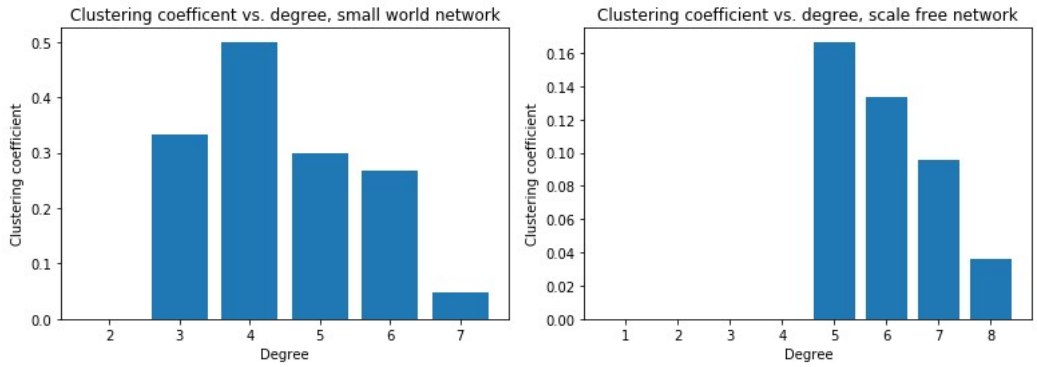


Figure 3.13. The clustering coefficient vs. degree for the two network types. Both indicate that the higher the node degree, the lower the clustering coefficient. This may be due to the higher degree nodes connecting to much lower ones rather than other more connected nodes, effectively reducing the number of closed triplets. See Appendix C for updated code. Source: Python.

Assortativity

The property of assortativity is defined as the likeliness of a network's nodes to be connected to others that are similar in some respects. It is common for the degree to be the similarity factor in network theory[16]. In reality, nodes often tend to connect with other nodes of similar degree; socialites are more likely to associate with other, well connected, socialites. Contrasting this with computer and biological networks, dissassortativity is apparent[17]. This could be due to the more distributed nature of a node's degree.

Simple assortativity can be demonstrated by changing the assortativity factor in the scale free network. The effects can be visualised as a NetworkX graph.

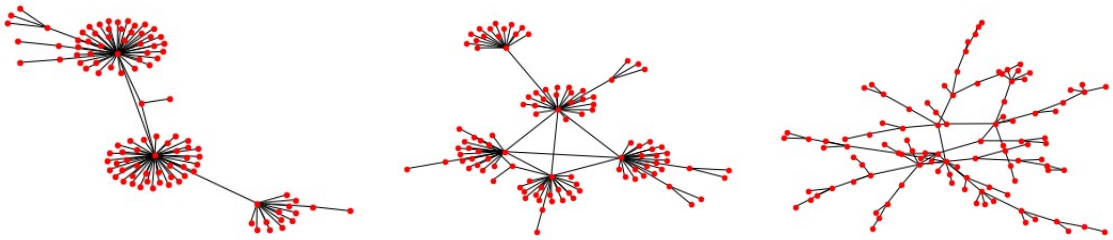


Figure 3.14. From left to right, an increase in the assortativity factor A. At the lowest values, the network contains few, densely connected hubs. At higher values, the number of hubs increases but the overall degree of the hubs is much lower. Source: NetworkX.

A method of measuring assortativity is by using the Pearson correlation coefficient[18] of degree between pairs of linked nodes. The assortativity coefficient for a network, r, is

(7)

$$r = \frac{\Sigma_{jk} jk(e_{jk} - q_j q_k)}{\sigma_q^2}$$

Here, q_k is the distribution of the remaining degree; the number of edges leaving a node other than the one that connects the pair, and e_{jk} is the joint probability distribution of the remaining degrees of the two nodes j and k.

The value r is always such that $1- < r < +1$. When $r = 1$, the network is completely assortative. When $r = -1$, the network is said to be disassortative.

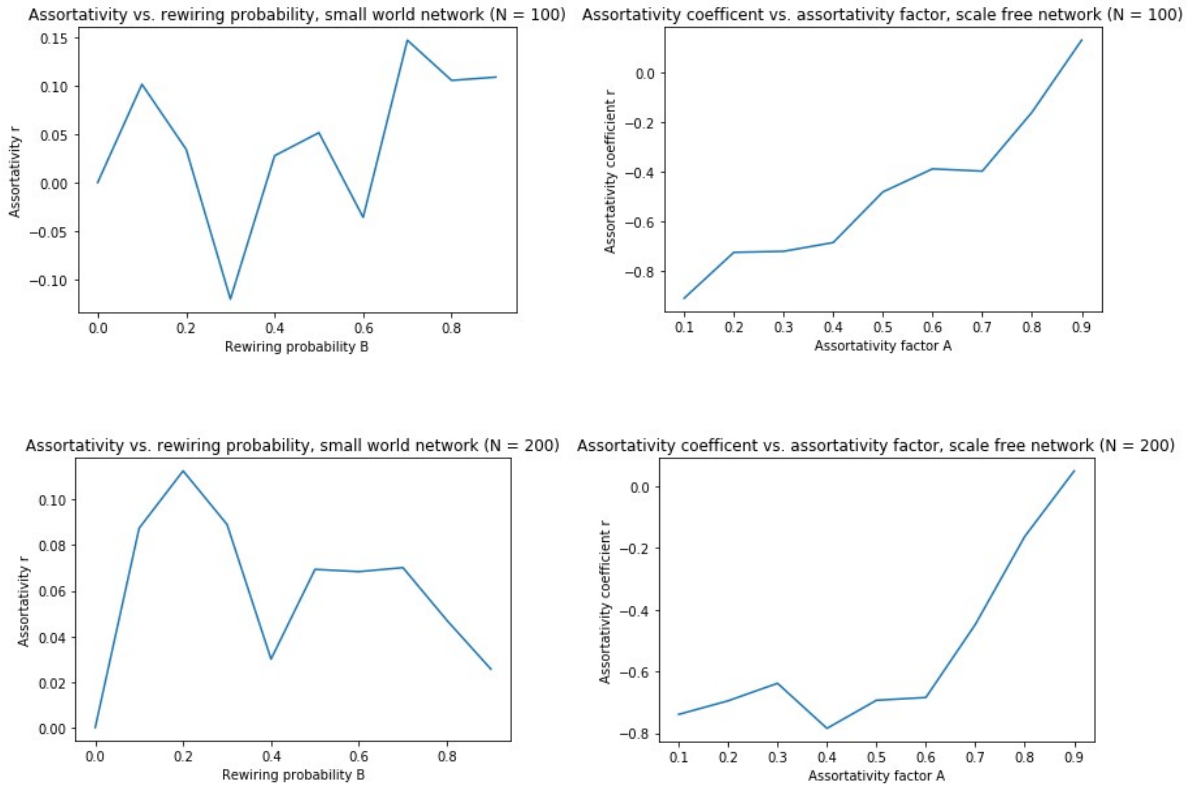


Figure 3.15. Pictured are the two networks for N = 100 (top) and N = 200 (bottom) nodes. Assortativity coefficient investigated for the defining characteristics of each network. There seems to be no apparent correlation between assortativity and rewiring probability B in the small world network, however there is a positive correlation with the assortativity coefficient A in the scale free – the graph begins highly disassortative and then ascends to become more assortative. This implies that A is a good factor for deciding a network's assortativity. Source: Python.

Robustness

The robustness of a network is how resilient the network is to removal of nodes or links. Does the removal of a node or edge at random disconnect a group of nodes from the network? How about the removal of a high degree node? In this section, the consequences of node removal (part of a wider study known as **percolation theory**) will be investigated, first for a random 'attack' and secondly for a targeted one.

Robustness is an important factor for a wide variety of fields and can provide insights to the study of diseases[19], the stability of financial systems[20] and the infrastructure of power grids[3]. In this study, effects of node removal will be investigated according to the results of the network degree distribution and connectivity of node degree.

Connectivity is a measure of the minimum number of nodes or edges required to be removed to disconnect remaining nodes into subgraphs. This results in isolating parts of the network from each other and is important for studying the properties of robustness. If an attack has already occurred on a network, it may be more vulnerable to future attacks.

The two attack functions, *do_random_attack* and *do_target_attack* (see Appendix A.3) remove, upon each iteration, a node from the input adjacency matrix (this is achieved by deleting a column and its corresponding row, then reordering the network) at random or upon highest degree respectively. It should be made clear that the targeted attack function removes, procedurally upon each iteration, the node of highest degree within the network. The effects of each function will now be explored.

For the scale free network, the random and targeted attacks were evaluated for varying values of N and A.

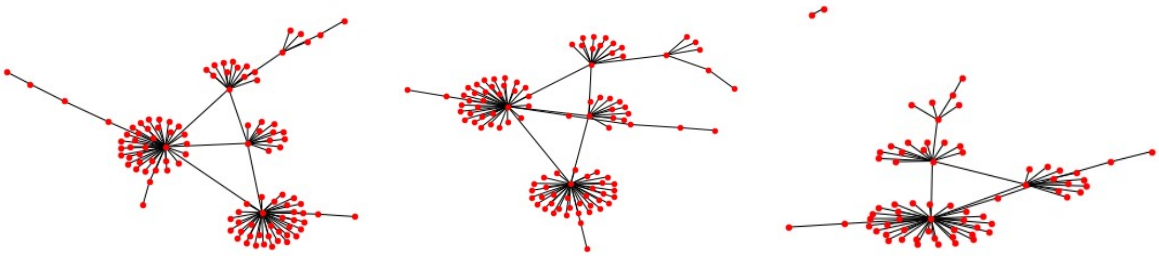
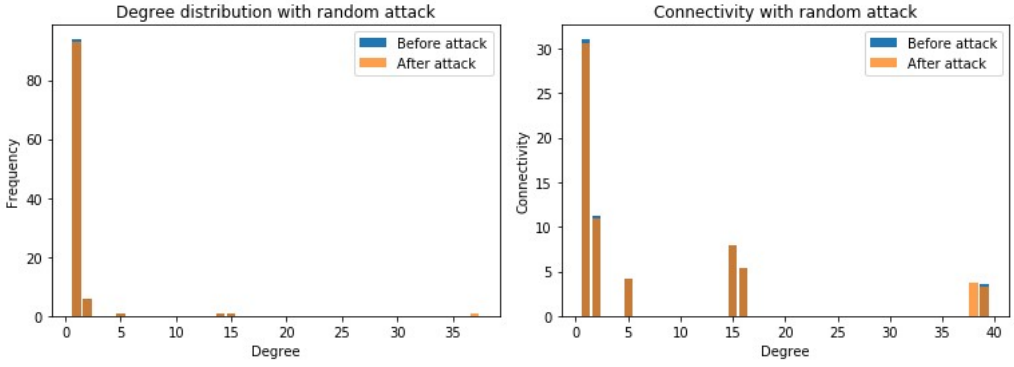


Figure 3.16. Network topology for N = 100 nodes and A = 0.2 before (left) and after a random (centre) and targeted attack (right). Source: NetworkX.



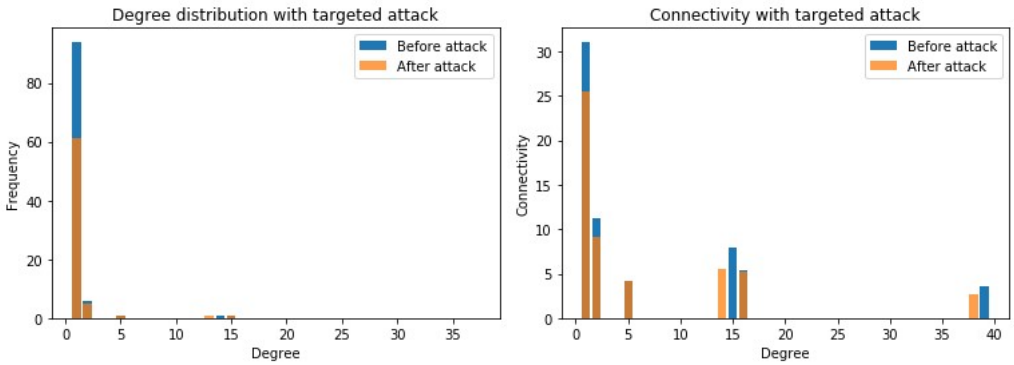


Figure 3.17. Degree distribution and connectivity compared before and after a random and targeted attack for the network in Figure 3.16. Source: NetworkX.

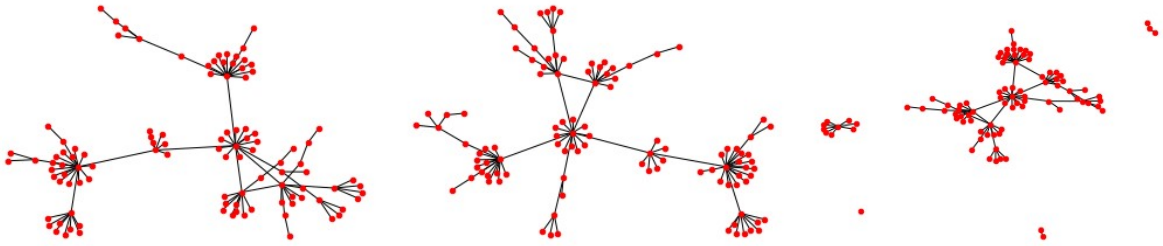


Figure 3.18. Network topology for N = 100 nodes and A = 0.5 before (left) and after a random (centre) and targeted attack (right). Source: NetworkX.

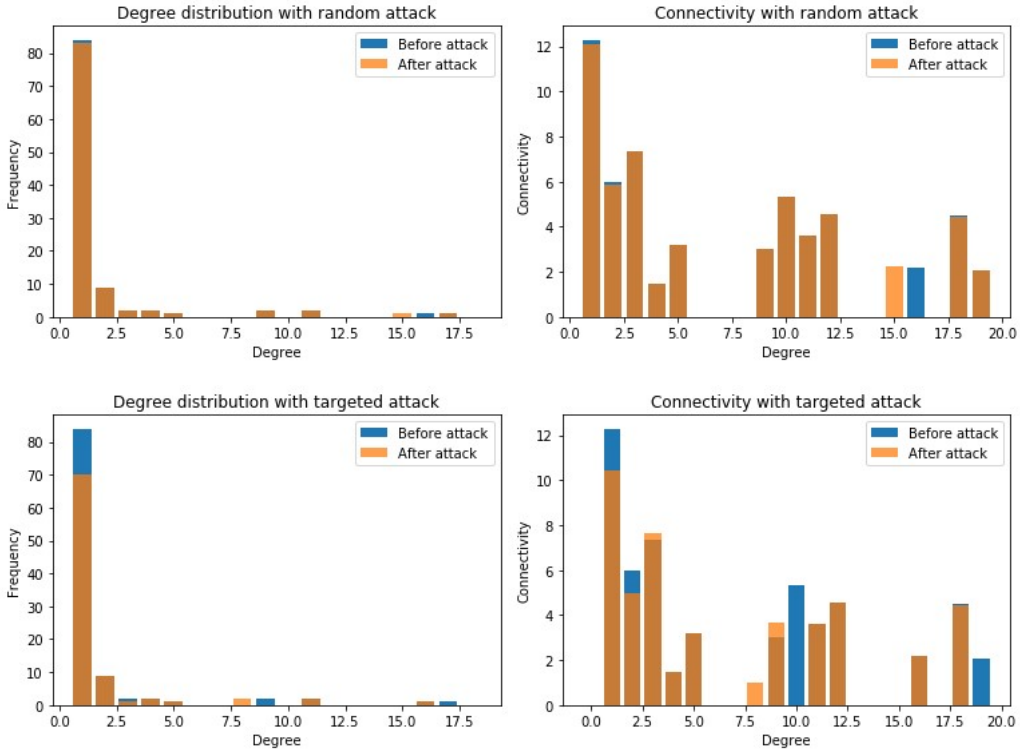


Figure 3.19. Degree distribution and connectivity compared before and after a random and targeted attack for the network in Figure 3.18. Source: NetworkX.

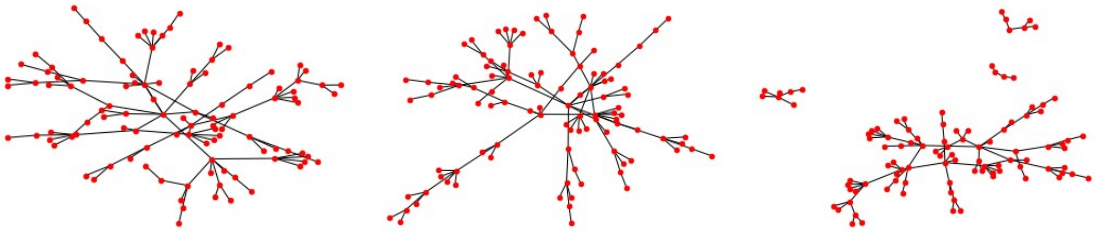


Figure 3.20. Network topology for N = 100 nodes and A = 0.8 before (left) and after a random (centre) and targeted attack (right). Source: NetworkX.

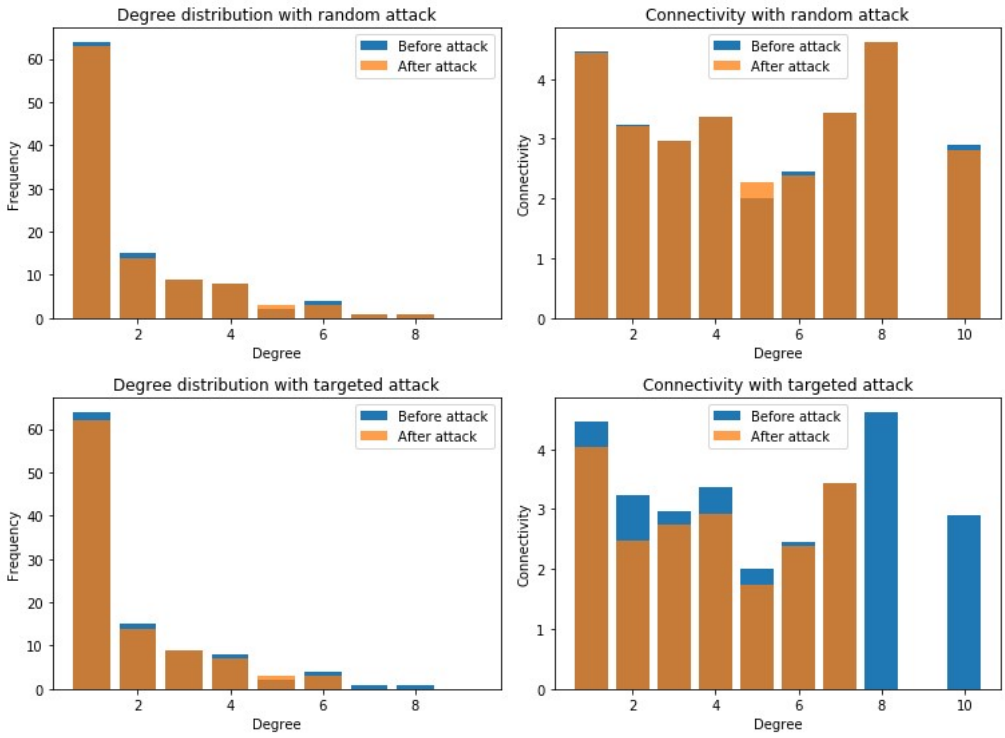


Figure 3.21. Degree distribution and connectivity compared before and after a random and targeted attack for the network in Figure 3.20. Source: NetworkX.

From these results, the scale free network appears to be resilient against random attacks but is vulnerable to targeted attacks. This may be due to the degree distribution; the removal of a random node may have little effect however disconnecting a hub may disconnect large parts of the network. The assortativity factor appears to have little effect, as the results appear fairly consistent between graphs.

For the small world network, the results are also evaluated for the same effect.

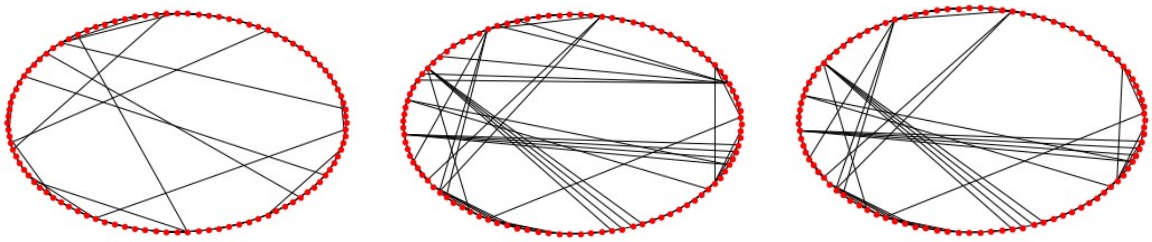


Figure 3.22. Network topology for N = 100 nodes and B = 0.2 before (left) and after a random (centre) and targeted attack (right). Source: NetworkX.

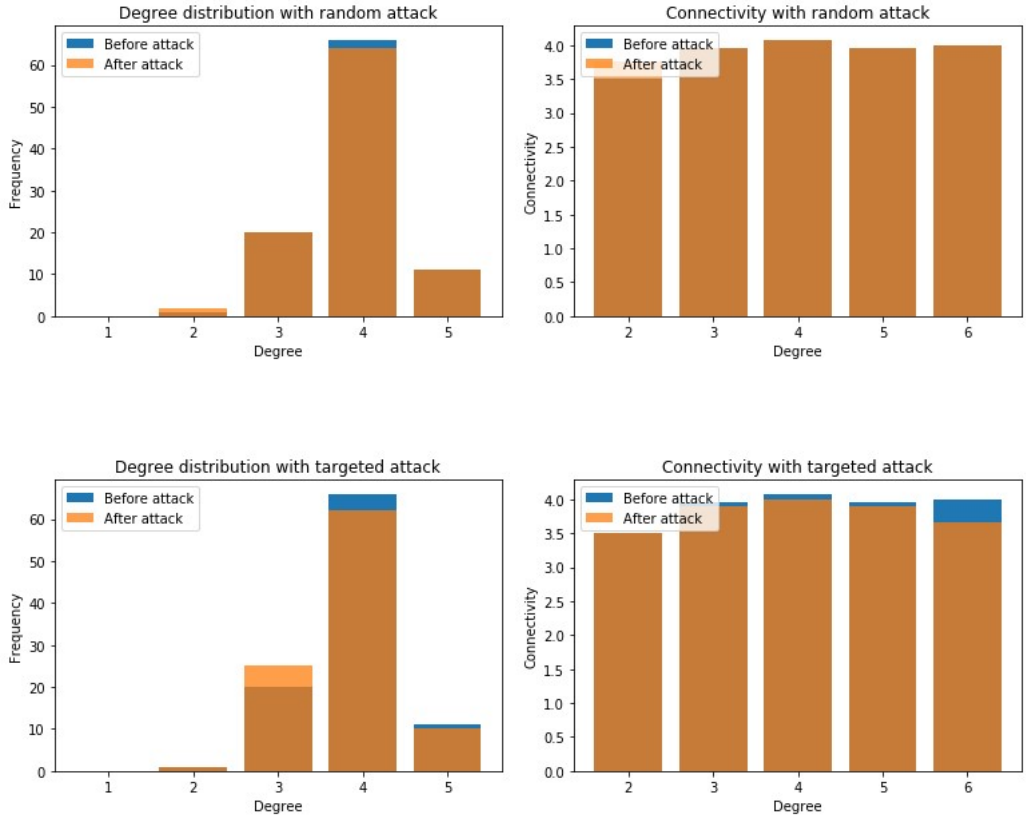


Figure 3.23. Degree distribution and connectivity compared before and after a random and targeted attack for the network in Figure 3.22. Source: NetworkX.

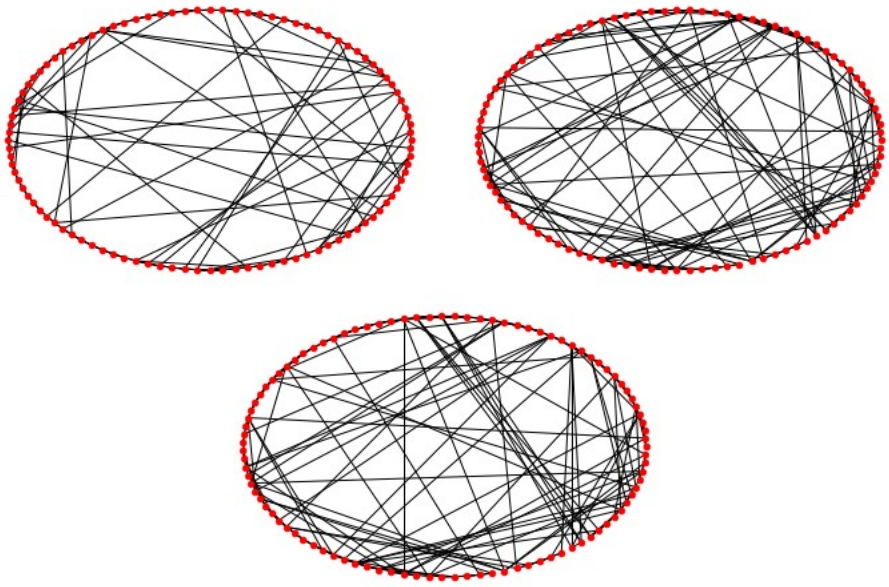


Figure 3.24. Network topology for N = 100 nodes and B = 0.5 before (left) and after a random (centre) and targeted attack (right). Source: NetworkX.

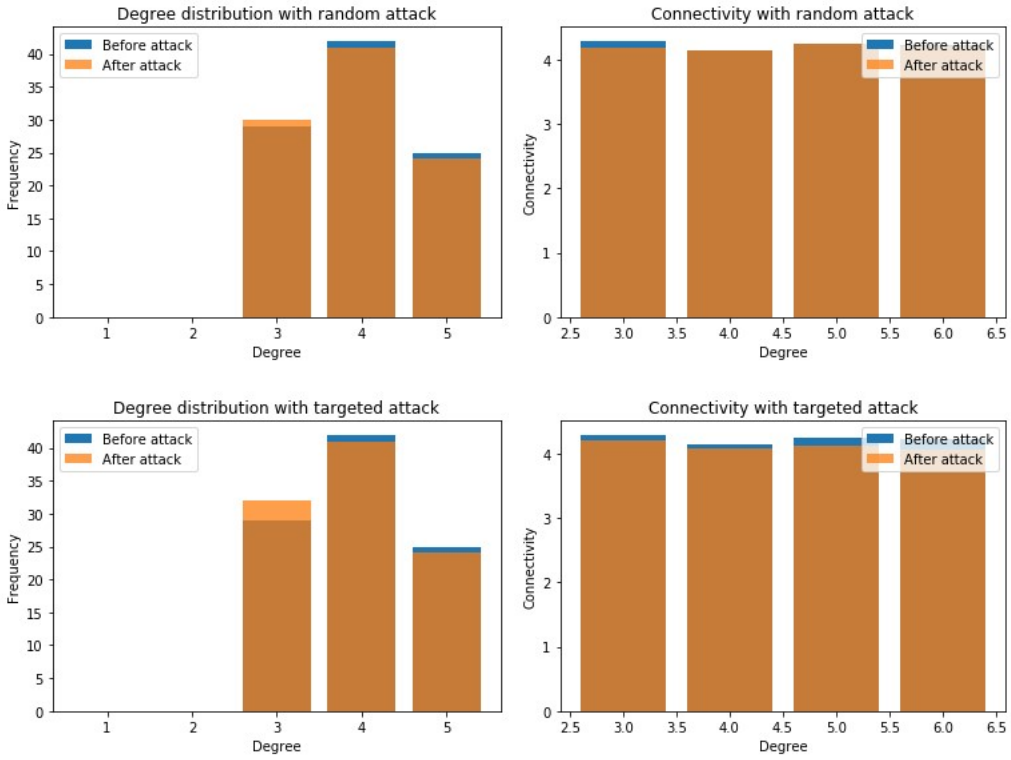


Figure 3.25. Degree distribution and connectivity compared before and after a random and targeted attack for the network in Figure 3.24. Source: NetworkX.

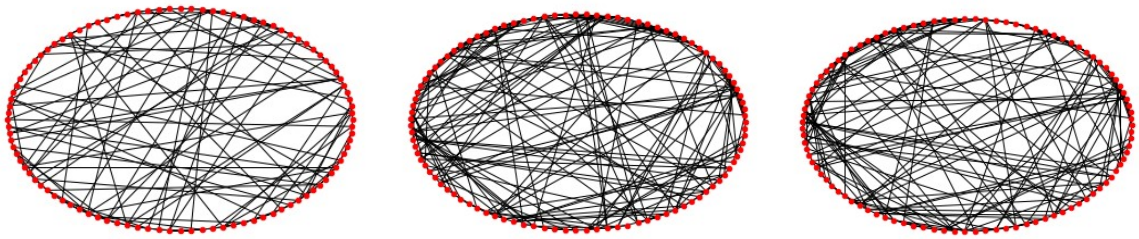


Figure 3.26. Network topology for N = 100 nodes and B = 0.8 before (left) and after a random (centre) and targeted attack (right). Source: NetworkX.

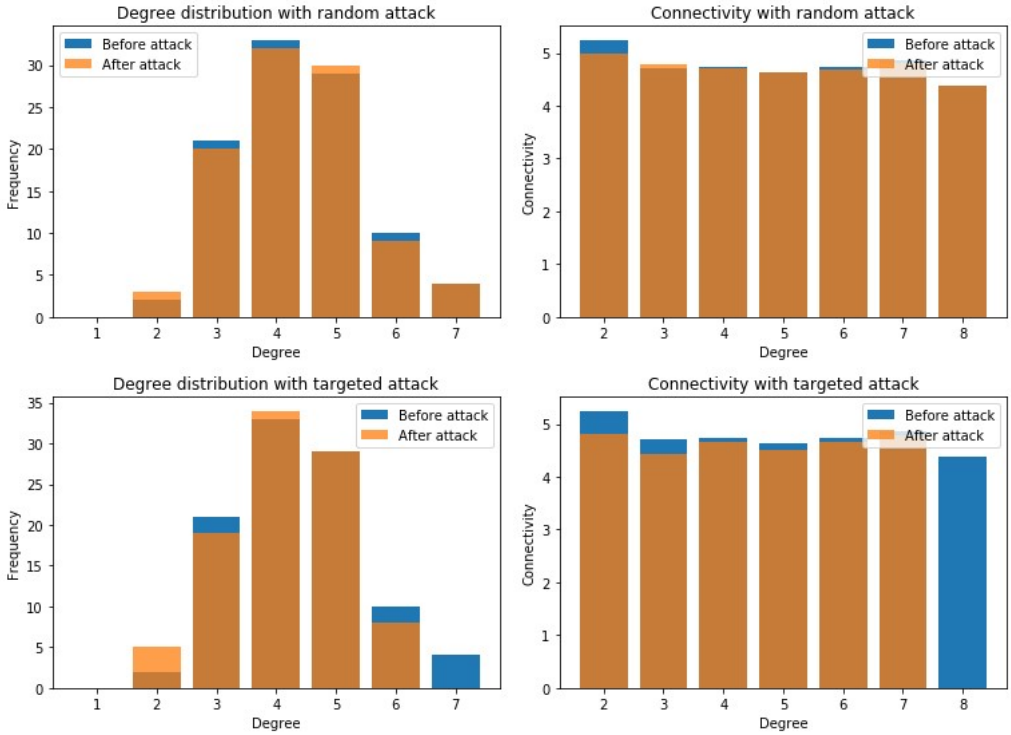


Figure 3.27. Degree distribution and connectivity compared before and after a random and targeted attack for the network in Figure 3.26. Source: NetworkX.

The small world network appears to be as susceptible to a targeted attack as the scale free, with little variation in connectivity and degree distribution after a random attack. This is due to the graph becoming more random as the small world effect takes place as there is a higher level of degree distribution. In a simple ring network containing nodes of similar orders, one would expect the targeted attack to have the same effect as a random one – removing the highest degree node in the network would have little effect because the majority (if not all) nodes in the network would have the same degree. Also, had the graphs in Figure 3.27 not been generated, the effects of the attacks would not have been as clear and the similarity with the scale free network unapparent from the visual illustration in Figure 3.26. This demonstrates the importance of degree distribution and connectivity analysis within network attacks.

In summary, both networks are resilient to random attacks but fall vulnerable to targeted attacks. If either of the types represented a data transmission network, it would make sense to anonymise the hubs within the network. This would reduce the consequences an attack such as DDoS[21] would have on the network as a whole.

4 Applications

Complex networks in the real world

Complex networks can be used to model many data sets in science and society. By understanding the correlations between their properties, they can be analysed and classified into an approximation of the small world or scale free networks.

One interesting application of complex networks is the study social networks using real world data. Social circles from the popular networking site, Facebook, can be input into a complex network analysis program to gain a better understanding of the links between individuals and their mutual friends.

Many real-world complex networks have thousands, if not millions of data points that would require a considerable amount of time and processing power to compute. Visualising a network as a graph may omit vital details as the scale factor for drawing a graph exceeds the resolution of the program.

The social network

The program *analyse.py* (Appendix C.2) takes anonymised data of social circles in Facebook [22] and the data can be visualised as a NetworkX graph.

Figure 4.1. Real Facebook data containing anonymised social ties between users on the platform. There are 4,039 nodes and 88,234 edges. Although the majority of the data cannot be represented individually, large clusters of highly connected individuals appear on the graph for the majority. The outliers existing on the individual strands could be fake Facebook accounts or less-connected individuals. Source: NetworkX.

Evaluating the network for its properties, using the software mentioned in the last section, generates some interesting correlations.

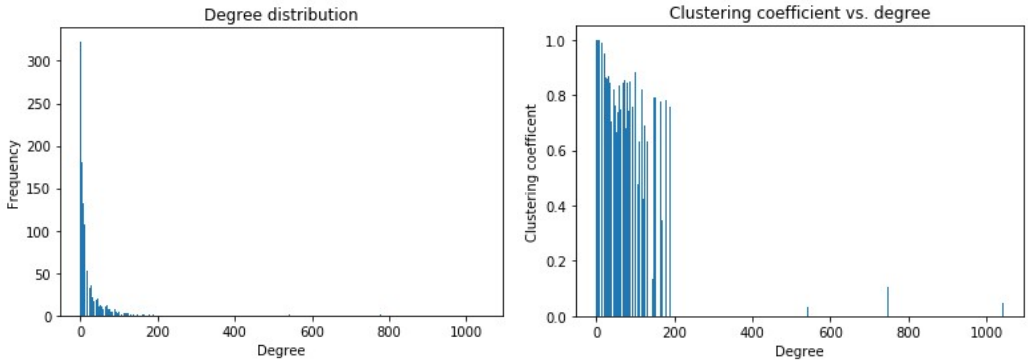


Figure 4.2. Degree distribution and clustering coefficient for the network nodes. There is an unsurprisingly high degree distribution within the network, which could represent the networking of well-connected individuals follows a power-law distribution. Such is the case for celebrities and social butterflies. The clustering coefficient for degree distribution appears to be high for lower-degree nodes: this may be interpreted by the tendency of less mainstream individuals to remain safely within their cliques. Source: Python.

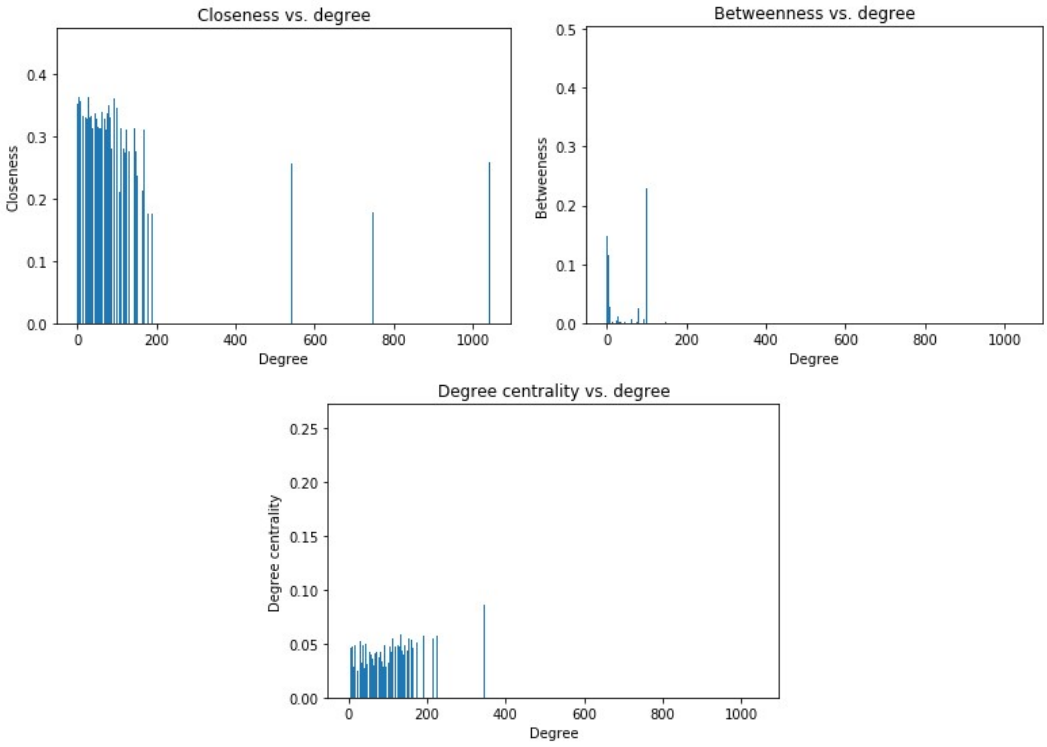


Figure 4.3. The three measures of centrality. The closeness data shows the that the majority of the people within the network can reach others with little steps required. Betweenness also suggests that a majority of people are a link for others between each other, however popular individuals are seldom a medium to reach others but rather the subject of the link. Degree centrality implies the majority of the less-linked individuals are more subject to flow of whatever is travelling through the network than the higher-linked ones. Source: Python.

The mean path length within the network L was found to be 3.6933 (to 5 s.f.) and can be interpreted as the number of individuals one person must connect with in order to reach another person at random. The assortativity coefficient r was 0.062875 (to 5 s.f.) implying the network is slightly assortative; individuals are more likely to network with other well-connected individuals than outsiders.

The network can be tested for robustness by performing random and targeted attacks upon it. Random attacks could represent any user being disconnected from a network (i.e. banned from Facebook or ostracised by their mutual friends) whilst targeted attacks affect well-connected individuals (social butterflies or celebrities) and may represent account closure, the fall in popularity of an individual or simply their choosing of another social networking platform and becoming inactive in the current. There are many interpretations of robustness in the social sector especially when the data is anonymised, as the key attributes are not recorded.

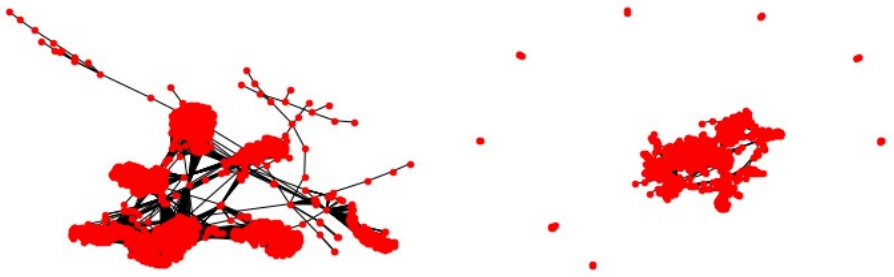


Figure 4.4. The same social network after a random attack of 5 nodes and a targeted attack of 5 high-degree nodes. The latter is far more effective at isolating people from each other than randomly removing people from the network. Source: NetworkX.

The measures of robustness following the two attacks follow, including connectivity and degree distribution before and after the attacks.

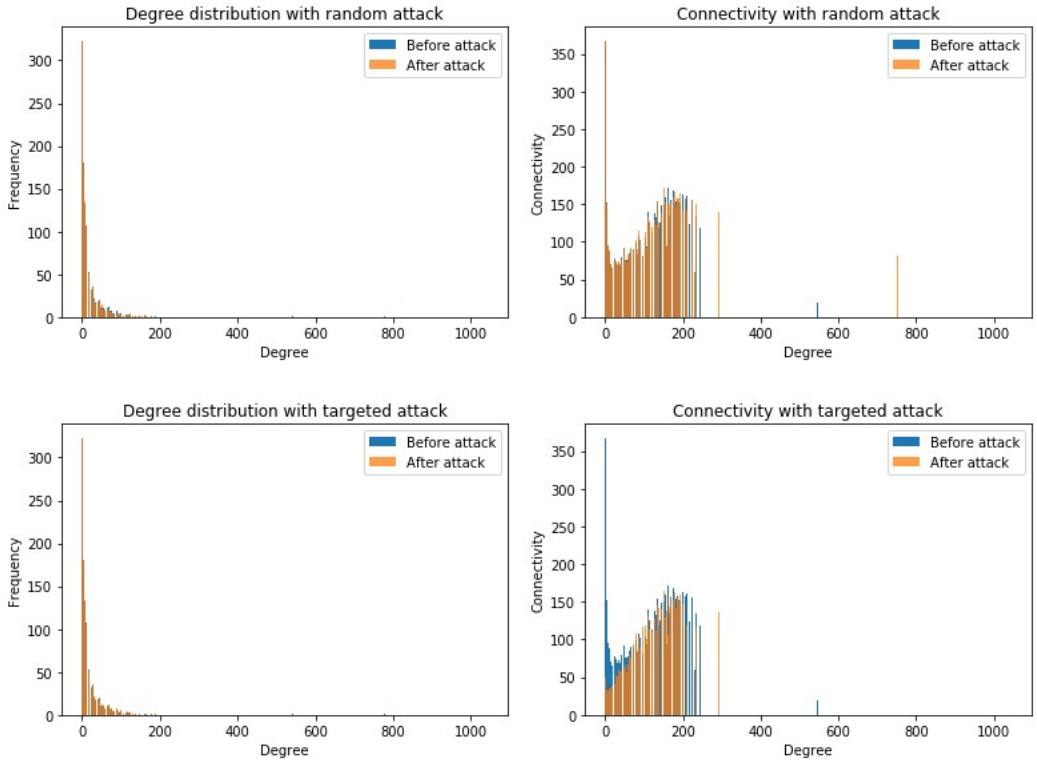


Figure 4.5. The results demonstrate that although degree distribution does not change much between the two attacks, connectivity changes drastically for the targeted attack. This implies that after one attack, successive attacks may have an even greater (and possibly exponential) impact on disconnecting the network. Thus, targeted attacks by node degree prove a very effective means of isolating a network. The behaviour of the lowest degree nodes reduced connectivity implies this network is similar to a scale free one.

5 Conclusion

Summary of findings

As a whole, this study has introduced the concept of networks, the factors that make networks complex, and an investigation into their properties and effects of attacks. Both the scale free and small world networks are interesting abstractions of real networks, however many real networks appear of a more random nature. The idea of classifying a network based on its attributes is one that is entirely up to the discretion of those who study it. Despite the theory that networks are deterministic in nature, one must conduct an in-depth study of all aspects in order to gain a comprehensive understanding.

Scale free networks illustrated traits similar to that of the small world ones, with an average distance between nodes having a proportional logarithmic relationship with the number of nodes on a network. As the rewiring probability increased, small world networks converged towards becoming more of a random network. Scale free networks became ring-like in nature as their assortativity increased; hubs became indistinguishable from each other as the very meaning of a hub became blurred with a decrease in average nodes per assortativity iteration, while the average mean path length between nodes increased significantly.

The social network investigated in the previous section is closer to that of a scale free network than a small world one. The similarities in degree distribution, centrality and robustness results support this theory. It could be said that this reflects the true nature of social networks in modern society; the few popular or well-connected individuals yield connections to several much less-known people, such as a one-to-many relationship. However, to a much larger scale over time, could this network have possibly converged to that of a small world one once the mutual friends with in it had become acquainted with each other more? This network was merely a snapshot of a network that, in nature, would be very dynamic and subject to change. People are becoming more connected every day, with ‘friendships’ changing throughout their lives.

The future

The field of Complex Networks is a new and widely evolving study that is becoming more relevant every day. Data science relies heavily on the understanding of many connected systems and gaining a strong mathematical understanding of the properties and behaviours of these networks can simplify difficult problems in ways more efficient than realised.

References

[1] Chris Budd, “Maths Goes Underground”, Mathematics Today, 49:5 (2013), 198–199

[2] Encyclopaedia of Physics (2nd Edition), R.G. Lerner, G.L. Trigg, VHC publishers, 1991, ISBN (Verlagsgesellschaft) 3-527-26954-1, ISBN (VHC Inc.) 0-89573-752-3

[3] Albert, R.; Albert, I.; Nakarado, G.L. (2004). "Structural Vulnerability of the North American Power Grid". Phys. Rev. E. 69 (2): 025103. arXiv:cond-mat/0401084. Bibcode:2004PhRvE..69b5103A. doi:10.1103/physreve.69.025103. PMID 14995510

[4] Choromański, K.; Matuszak, M.; MięKisz, J. (2013). "Scale-Free Graph with Preferential Attachment and Evolving Internal Vertex Structure". Journal of Statistical Physics. 151 (6): 1175–1183. Bibcode:2013JSP...151.1175C. doi:10.1007/s10955-013-0749-1

[5] M. E. J. Newman, “The Structure and Function of Complex Networks”, Department of Physics, University of Michigan (2003), p30

[6] Albert-László Barabási, Réka Albert, Hawoong Jeong, “Scale-free characteristics of random networks: the topology of the world-wide web”, Department of Physics, College of Science, 225 Nieuwland Science Hall, University of Notre-Dame (2000). https://doi.org/10.1016/S0378-4371(00)00018-2

[7] S. H. Strogatz, D. J. Watts (1998). "Collective dynamics of 'small-world' networks". Nature. 393(6684): 440– 442

[8] Milgram, Stanley (May 1967). "The Small World Problem". Psychology Today. Ziff-Davis Publishing Company

[9] Andreas I. Reppas, Konstantinos Spiliotis, Constantinos I. Siettos (2015). “Tuning the average path length of complex networks and its influence to the emergent dynamics of the majority-rule model”. Mathematics and Computers in Simulation, 109, March 2015, Pages 186–196.

[10] V. R. Sole; M. M. Jose (2001). "Complexity and fragility in ecological net-works". Proc. R. Soc. Lond. B. 268 (1480): 2039–45. arXiv:cond-mat/0011196. doi:10.1098/rspb.2001.1767. PMC 1088846. PMID 11571051

[11] Newman, M.E.J. 2010. Networks: An Introduction. Oxford, UK: Oxford University Press.

[12] Krackhardt, David (June 1990). "Assessing the Political Landscape: Structure, Cognition, and Power in Organizations". Administrative Science Quarterly. 35 (2): 342–369. doi:10.2307/2393394. JSTOR 2393394

[13] Sabidussi, G (1966). "The centrality index of a graph". Psychometrika. 31 (4): 581–603. doi:10.1007/bf02289527. hdl:10338.dmlcz/101401. PMID 5232444

[14] Freeman, Linton (1977). "A set of measures of centrality based upon betweenness". Sociometry. 40 (1): 35–41. doi:10.2307/3033543. JSTOR 3033543

[15] Brandes, Ulrik (2001). "A faster algorithm for betweenness centrality" (PDF). Journal of Mathematical Sociology. 25 (2): 163–177. CiteSeerX 10.1.1.11.2024. doi:10.1080/0022250x.2001.9990249.

[16] Newman, M. E. J. (27 February 2003). "Mixing patterns in networks". Physical Review E. American Physical Society (APS). 67 (2): 026126. arXiv:cond-mat/0209450. Bibcode:2003PhRvE..67b6126N. doi:10.1103/physreve.67.026126. ISSN 1063-651X

[17] Newman, M. E. J. (28 October 2002). "Assortative Mixing in Networks". Physical Review Letters. American Physical Society (APS). 89 (20): 208701. arXiv:cond-mat/0205405. doi:10.1103/physrevlett.89.208701. ISSN 0031-9007. PMID 12443515

[18] "SPSS Tutorials: Pearson Correlation", Kent State University. Accessed 7th May 2020.

[19] A. Motter; N. Gulbahce; E. Almaas & A.-L. Barabási (2008). "Predicting synthetic rescues in metabolic networks". Molecular Systems Biology. 4: 1–10. arXiv:0803.0962. doi:10.1038/msb.2008.1. PMC 2267730. PMID 18277384.

[20] Haldane, A. G.; May, R. M. (2011). "Systemic risk in banking ecosystems". Nature. 469 (7330): 351–355. Bibcode:2011Natur.469..351H. doi:10.1038/nature09659. PMID 21248842

[21] "Understanding Denial-of-Service Attacks". US-CERT. 6 February 2013. Accesssed 7 May 2020.

[22] J. McAuley and J. Leskovec. Learning to Discover Social Circles in Ego Networks, Stanford University. NIPS, 2012

Appendices

Appendix A

This section includes the program *random_network.py* which involves the creation of a random network as an adjacency matrix and the drawing of it in a graph. The program *network_functionality.py* is also included and contains essential functionality for obtaining the properties of a network and analysing them.

```
import numpy as np
import random
import networkx as nx
import matplotlib.pyplot as plt

N = 8 # total nodes
P = 0.2 # universal probability of connection
max_weight = 10 # max weight of nodes

# create adjacency matrix
adj_matrix = np.zeros((N,N), dtype=np.int)
# create graph using NetworkX
G = nx.Graph()
G.add_nodes_from([0, N - 1])

# populate adjacency matrix
# connections according to P
for i in range(N):
    for j in range(N):
        # random connection chance
        # measured against P
        # to decide if connection or not
        rand = random.uniform(0, 1)
        if rand <= P:
            adj_matrix[i,j] = random.uniform(1, max_weight) # assign random non-zero weight
            G.add_edge(i, j, weight=adj_matrix[i,j])
        # At some point might want to account for conjugate pair j,i too

# draw network using NetworkX
# draw nodes with labels
pos=nx.spring_layout(G)
nx.draw_networkx_nodes(G, pos, labels=adj_matrix,node_color='r', node_size=300, alpha=1)

# draw arcs
nx.draw_networkx_edges(G, pos, labels=adj_matrix,node_color='r', alpha=1)

# draw labels for nodes and weights of arcs
nx.draw_networkx_edge_labels(G, pos, nx.get_edge_attributes(G,'weight'))
nx.draw_networkx_labels(G, pos,
                        labels=None,
                        font_size=12,
                        font_color='k',
                        font_family='sans-serif',
                        font_weight='normal',
                        alpha=1.0,
                        bbox=None,
                        ax=None)

plt.savefig("simple_path.png") # save as png
plt.axis('off')
plt.show() # display
```

A.1. The python program random_network.py. An implementation of a simple random network in Python, from defining the network as an adjacency matrix to drawing it with the NetworkX library. Source: Python.

```
"""
----------------------------------------
Standard functions
----------------------------------------
"""

# Creates a graph useful for visualisation
def create_graph(adj_matrix, G, circular):
    N = len(adj_matrix)
    G.add_nodes_from([0, N-1])
    for i in range (N):
        for j in range(N):
            if adj_matrix[i,j] == 1:
                G.add_edge(i,j, weight=1)
            if adj_matrix[i,j] == 0 and G.has_edge(i,j):
                G.remove_edge(i,j)
                if not nx.is_connected(G):
                    G.add_edge(i,j)
    if circular == True:
        pos=nx.circular_layout(G)
    else:
        pos=nx.spring_layout(G)
    nx.draw_networkx_nodes(G, pos, labels=adj_matrix,node_color='r', node_size=20, alpha=1)
    nx.draw_networkx_edges(G, pos, labels=adj_matrix,node_color='r', alpha=1)
    plt.axis('off')
    plt.show()

# Checks if node is connected to any other nodes
# Returns true or false
def check_is_node_connected(adj_matrix, node):
    N = len(adj_matrix)
    if sum(adj_matrix[node]) == 0:
        return False
    else:
        return True

# Returns array of degree and probabilities of nodes
def get_degree_info(adj_matrix):
    N = len(adj_matrix)
    prob_arr    = np.zeros((N), dtype=np.int)
    degree_arr  = np.zeros((N), dtype=np.int)
    for i in range(0,N):
        for j in range(0,N):
            if adj_matrix[i,j] == 1:
                degree_arr[i] += 1
    total_deg = sum(degree_arr)
    prob_arr = degree_arr/total_deg
    return degree_arr, prob_arr

# Returns degree distribution of frequency vs. node
def get_degree_distribution(adj_matrix):
    deg_arr  = get_degree_info(adj_matrix)[0]
    max_deg  = max(deg_arr)
    deg_axis = np.arange(0,max_deg)
    frq_axis = np.zeros(max_deg)
    for i in range (0, max_deg):
        for j in range(len(deg_arr)):
            if deg_arr[j] == i:
                frq_axis[i] += 1
    return deg_axis, frq_axis

# Returns mean path length of network
def get_mean_path_len(adj_matrix):
    # Create distance matrix of nodes from eachother
    # Numbers represent nodes totraverse
    N = len(adj_matrix)
    shortest_path = scipy.sparse.csgraph.shortest_path(adj_matrix,
                                                       method='auto',
                                                       directed=False,
                                                       return_predecessors=False,
                                                       unweighted=True,
                                                       overwrite=False)
    # In the scipy function, non-connections are represented by infinite values
    # Represent non-connections as zeroes instead to avoid errors
    for i in range(N):
        for j in range(N):
            if shortest_path[i, j] == np.inf:
                shortest_path[i, j] = 0
    L = (1/(N*(N-1)))*sum(sum(shortest_path))
    return L
```

A.2. The python program network_functionality.py. Standard functions include creating a graph in NetworkX from an adjacency matrix, checking if a node is connected to the network, obtaining the degree information for each node, obtaining the degree distribution and finding the mean path length. These functions will be re-used throughout the study, hence containing their own file. Source: Python.

```
"""
----------------------------------------
NetworkX functions
----------------------------------------
"""

# As a safe measure, I decided to use the NetworkX library for these functions
# When trying to construct them manually there were data errors from calculated numbers being out of range
# The functionality is the same and an explanation of how they work will be in the report

# Return betweenenss centrality per node
def get_betweenness_per_node(G):
    return nx.betweenness_centrality(G, k=None, normalized=True, weight=None, endpoints=False, seed=None)

# Return degree centrality per node
def get_degree_centrality(G):
    return nx.degree_centrality(G)

# Return clustering coefficient per node
def get_local_clustering_coeffs(G):
    return nx.clustering(G, nodes=None, weight=None)

# Return degree assortativity of graph
def get_degree_assortativity(G):
    return nx.degree_assortativity_coefficient(G)

def get_closeness_per_node(G):
    return nx.closeness_centrality(G, u=None, distance=None, wf_improved=True)

def get_eigencen_per_node(G):
    return nx.eigenvector_centrality(G, max_iter=100, tol=1e-06, nstart=None, weight=None)

"""
----------------------------------------
Attack functions
----------------------------------------
"""

# Returns adjacency matrix with removed nodes
# X = how many nodes to remove
def do_random_attack(adj_matrix,x):
    # Each deletion iteration
    for i in range(x):
        N = len(adj_matrix)
        node = random.randint(0,N-1)
        # Delete row
        adj_matrix = np.delete(adj_matrix, (node), axis=0)
        # Delete column
        adj_matrix = np.delete(adj_matrix, (node), axis=1)
    return adj_matrix

# Returns adjacencty matrix with removed highest order node
def do_target_attack(adj_matrix, x):
    # Each deletion iteration
    for i in range(x):
        # Attack the highest degree node
        N = len(adj_matrix)

        # Find highest degree node
        degree_arr = get_degree_info(adj_matrix)[0]
        target = degree_arr.argmax()

        # Delete row
        adj_matrix = np.delete(adj_matrix, (target), axis=0)
        # Delete column
        adj_matrix = np.delete(adj_matrix, (target), axis=1)
    return adj_matrix
```

A.3. The NetworkX functions extract the relevant properties from the graph constructed from the adjacency matrix. Preference for NetworkX functions was chosen due to complications with creating the functions. Some of these are deprecated as a more rigorous approach is included (Appendix C). Followed are the attack functions, a random and targeted attack that can be performed on a network x amount of times. These functions remove a node from the network upon each iteration. Source: Python.

```
"""
---------------------------------------
Data entry functions
---------------------------------------
"""

# These functions arrange data into a tabulated format
# This data can be used to draw graphs
# Skip zeroth element in most due to it containing meaningless data
# Returns tables of x, y

def tab_degree_dist(adj_matrix):
    data = get_degree_distribution(adj_matrix)
    x_axis = data[0]
    y_axis = data[1]
    return x_axis[1:], y_axis[1:]

def tab_betweenness_vs_degree(G, adj_matrix):
    data = list(get_betweenness_per_node(G).values())
    N = len(adj_matrix)
    x_axis = get_degree_info(adj_matrix)[0]
    y_axis = np.zeros(N)
    for i in range(N):
        y_axis[i] = data[i]
    return x_axis, y_axis

def tab_closeness_vs_degree(G, adj_matrix):
    data = list(get_closeness_per_node(G).values())
    N = len(adj_matrix)
    x_axis = get_degree_info(adj_matrix)[0]
    y_axis = np.zeros(N)
    for i in range(N):
        y_axis[i] = data[i]
    return x_axis, y_axis

def tab_degree_centrality_vs_degree(G, adj_matrix):
    data = get_degree_centrality(G)
    N = len(data)
    x_axis = get_degree_info(adj_matrix)[0]
    y_axis = np.zeros(N)
    for i in range(N):
        y_axis[i] = data[i]
    return x_axis, y_axis

def tab_eigencen_vs_degree(G, adj_matrix):
    data = get_eigencen_per_node(G)
    N = len(data)
    x_axis = get_degree_info(adj_matrix)[0]
    y_axis = np.zeros(N)
    for i in range(N):
        y_axis[i] = data[i]
    return x_axis, y_axis

def tab_clust_coeff_vs_degree(G, adj_matrix):
    data = get_local_clustering_coeffs(G)
    N = len(data)
    y_axis = np.zeros(N)
    x_axis = get_degree_info(adj_matrix)[0]
    for i in range(N):
        y_axis[i] = data[i]
    return x_axis, y_axis
```

A.4. These functions input a NetworkX graph and/or an adjacency matrix, pass the input to the relevant property extraction function and return tabulated data that can be plotted as a graph. This will be useful for visualisation of network properties later. Source: Python.

Appendix B

This section displays the programs written to create the scale free and small world networks. Includeed are two programs, *scale_free_network.py* (B.1) and *small_world_network.py* (B.2).

```
"""
----------------------------------------
Special functions
----------------------------------------
"""

# Returns a random network
def create_random_network(N,P):
    adj_matrix = np.zeros((N,N), dtype=np.int)
    for i in range(N):
        for j in range(N):
            rand = random.uniform(0, 1)
            if rand <= P:
                adj_matrix[i,j] = 1
                adj_matrix[j,i] = 1
    return adj_matrix

# Scale free network
# Barabási–Albert model
# Returns new adjacency matrix
def create_scale_free_network(old_matrix, S,A):
    N = len(old_matrix)
    for i in range(S + 1):
        new_dim    = N + i
        new_matrix = np.zeros((new_dim, new_dim), dtype=np.int)

        # Port old matrix to new matrix
        for j in range(0, new_dim - 1):
            for k in range(0, new_dim - 1):
                new_matrix[j,k] = old_matrix[j,k]
                new_matrix[k,j] = old_matrix[k,j]

        # Get information about node degree
        degree_info = get_degree_info(new_matrix)
        degree_arr  = degree_info[0]
        prob_arr    = degree_info[1]

        # Choose node to attach new node to
        random_trial = random.uniform(0, max(prob_arr)) * (1-A)
        high_nodes   = np.where(random_trial < prob_arr)[0]
        chosen_node  = random.choice(high_nodes)

        # Check node doesn't disconnect graph!
        while not (check_is_node_connected(new_matrix, chosen_node)):
            chosen_node  = random.choice(high_nodes)
            print(chosen_node)

        # Add chosen node to new
        new_matrix[chosen_node, new_dim - 1] = 1
        new_matrix[new_dim - 1, chosen_node] = 1


        # Prepare for next step
        old_matrix = new_matrix
    return new_matrix
```

```
"""
----------------------------------------
Main program
----------------------------------------
"""

N = 5 # Nodes in original network
P = 0.5 # Connectivity of original network
S = 200 # How many new nodes to add for scale free network
A = 0.9 # Assortativity factor; how likely new hubs are to be formed
S = 100 # How many new nodes to add
G = nx.Graph()
random_network = create_random_network(N,P)
scale_free_network = create_scale_free_network(random_network, S, A)
create_graph(scale_free_network,G,0)
```

B.1. The functions for generating a random and scale free network, and drawing it. N and P are defined later in the code as parameters for the initial number of nodes and connectivity of the original network. Program execution follows defining parameters N, P, A and S. Source: Python.

```
def create_ring_network(N):
    adj_matrix = np.zeros((N,N), dtype=np.int)
    # Connect to neighbours each side
    for i in range(N-1):
        for j in range(N-1):
            if i+1 == j+1:
                adj_matrix[i,j+1] = 1
                adj_matrix[i+1,j] = 1
                G.add_edge(i+1, j, weight=adj_matrix[i,j])
    # Connect to two neigbhours each side
    if N > 50:
        for i in range(N-2):
            for j in range(N-2):
                if i+2 == j+2:
                    adj_matrix[i,j+2] = 1
                    adj_matrix[i+2,j] = 1
                    G.add_edge(i+2, j, weight=adj_matrix[i,j])
    # Connect to three neighbours each side
    if N > 100:
        for i in range(N-3):
            for j in range(N-3):
                if i+3 == j+3:
                    adj_matrix[i,j+3] = 1
                    adj_matrix[i+3,j] = 1
                    G.add_edge(i+3, j, weight=adj_matrix[i,j])
    # Close the loop
    adj_matrix[0, N-1] = 1
    adj_matrix[N-1, 0] = 1
    return adj_matrix

def create_small_world_network(adj_matrix, B):
    # Watts-Strogatz model
    N = len(adj_matrix) # total number of nodes
    chosen_arcs_amount = int(B * N) # number of arcs chosen randomly

    for i in range(chosen_arcs_amount):
        rewired_node = int(random.uniform(0, N))
        adj_matrix[rewired_node, rewired_node - 1] = 0
        adj_matrix[rewired_node - 1, rewired_node] = 0
        # Check arc choice doesn't disconnect graph!
        while not (check_is_node_connected(adj_matrix, rewired_node)):
            rewired_node = int(random.uniform(0, N))
            adj_matrix[rewired_node, rewired_node - 1] = 0
            adj_matrix[rewired_node - 1, rewired_node] = 0
        random_node = int(random.uniform(0, N))
        adj_matrix[rewired_node, random_node] = 1
        adj_matrix[random_node, rewired_node] = 1

    return adj_matrix
```

```
"""
----------------------------------------
Main program
----------------------------------------
"""

# Initialise
N = 200
B = 0.1
G = nx.Graph()
ring_network = create_ring_network(N)
small_world_network = create_small_world_network(ring_network, B)
create_graph(small_world_network,G,1)
```

B.2. The functions for creating a ring network and evolving the small world effect upon it. Similar to the scale free network, the program is run with initial values of N (number of nodes on the ring network) and the rewiring probability B. Source: Python.

Appendix C

This section contains the miscellaneous functions used in *network_funcitonality.py* that overwrite the previous functions with more advanced functionality. The file *analyse.py* is also included demonstrating the input of real data into a python program and invoking the features in *network_functionality.py* to analyse it.

```
# Return clustering coefficient per node
def get_local_clustering_coeffs(adj_matrix):
    N = len(adj_matrix)
    C_arr = np.zeros(N)
    # For each coulmn i
    for i in range(N):
        v = adj_matrix[i]
        len_v = len(v)
        Kv = sum(v)
        Nv = 0

        v_neighbours = []

        # For each row j
        for j in range(len_v):
            if v[j] == 1:
                v_neighbours.append(j)

        # Go through each of the neighbours
        for k in range(len(v_neighbours)):
            # In pairs, check are they connected?
            # If so, increment Nv
            for l in range(len(v_neighbours)):
                node_1 = v_neighbours[k]
                node_2 = v_neighbours[l]
                if adj_matrix[node_1,node_2] == 1 or adj_matrix[node_2,node_1] == 1:
                    Nv += 1

        # Calculate the clustering coefficient
        if (Nv) != 0:
            C_arr[i] = (2*Nv)/(Kv*(Kv - 1))
        print(C_arr[i])

    return C_arr
```

C.1. The function get_local_clustering_coeffs is the implementation of (6) and returns the clustering coefficient for the input array. Source: Python.

```
"""
Created on Thu May  7 01:04:15 2020

@author: kavianshirkoohi
"""

from pylab import*
import scipy as scipy
import numpy as np
import random
import networkx as nx
import matplotlib.pyplot as plt
import pandas
from network_functionality import *

"""
-----------------------------------------
Special functions
-----------------------------------------
"""

# Read two column data from file separated by whitespace
def import_from_file(filename):
    data = (pandas.read_table(filename, delim_whitespace=True, skiprows=1)).to_numpy()
    data_arr = np.column_stack((data[:,0],data[:,1]))
    N = len(data_arr)
    return data_arr

# Port 2D data frame to adjacency matrix
# Each row represents an edge connection
# Returns adjacency matrix as an array
def port_data_to_matrix(data_arr):
    max_data = np.arange(0,2,1)
    max_data[0] = max(data_arr[:,0])
    max_data[1] = max(data_arr[:,1])
    size = max(max_data) + 1
    adj_matrix = [[0 for i in range(size)] for j in range(size)]
    for row,column in data_arr:
        adj_matrix[int(row)][int(column)] = 1
    return np.array(adj_matrix)

"""
-----------------------------------------
Main program
-----------------------------------------
"""

# Convert data frame to adjacency matrix and draw graph
# Graph will be used to apply NetworkX to analyse the properties

data_arr = import_from_file('facebook_combined.txt')
adj_matrix = port_data_to_matrix(data_arr)
G = nx.Graph()
create_graph(adj_matrix, G, 0)

"""
# Drawing graphs
plt.title('Degree distribution')
degree_dist = tab_degree_dist(adj_matrix)
plt.bar(degree_dist[0],degree_dist[1])
plt.ylabel('Frequency')
plt.xlabel('Degree')
plt.show()

plt.title('Clustering coefficient vs. degree')
clustering = tab_clust_coeff_vs_degree(G, adj_matrix)
```

C.2. The program analyse.py inputs data from a file containing columnal information of connections between data points. This data is ported to an adjacency matrix and analysed in the same manner as that of the scale free and small world networks. The following figures below are part of the same program. Notice some of the code has been commented out, to utilise different parts of the program. Source: Python.

```
plt.title('Clustering coefficient vs. degree')
clustering = tab_clust_coeff_vs_degree(G, adj_matrix)
plt.bar(clustering[0],clustering[1])
plt.ylabel('Clustering coefficent')
plt.xlabel('Degree')
plt.show()
"""

"""
plt.title('Betweenness vs. degree')
betweenness = tab_betweenness_vs_degree(G, adj_matrix)
plt.bar(betweenness[0],betweenness[1])
plt.ylabel('Betweeness')
plt.xlabel('Degree')
plt.show()

plt.title('Closeness vs. degree')
betweenness = tab_closeness_vs_degree(G, adj_matrix)
plt.bar(betweenness[0],betweenness[1])
plt.ylabel('Closeness')
plt.xlabel('Degree')
plt.show()

plt.title('Eigenvector centrality vs. degree')
betweenness = tab_eigencen_vs_degree(G, adj_matrix)
plt.bar(betweenness[0],betweenness[1])
plt.ylabel('Eigenvector centrality')
plt.xlabel('Degree')
plt.show()

plt.title('Degree centrality vs. degree')
centrality = tab_degree_centrality_vs_degree(G, adj_matrix)
plt.bar(centrality[0],centrality[1])
plt.ylabel('Degree centrality')
plt.xlabel('Degree')
plt.show()
"""

# Ascertaining properties

L = get_mean_path_len(adj_matrix)
print("\nMean path length, L =", L)

r = get_degree_assortativity(G)
print("\nAssortativity coefficient, r =", r)

# Random attack
random_attack = do_random_attack(adj_matrix,5)
H = nx.Graph()
create_graph(random_attack, H, 0)

degree_dist = tab_degree_dist(adj_matrix)
line1 = plt.bar(degree_dist[0],degree_dist[1])
degree_dist = tab_degree_dist(random_attack)
line2 = plt.bar(degree_dist[0],degree_dist[1], alpha=0.75)
plt.title('Degree distribution with random attack')
plt.ylabel('Frequency')
plt.xlabel('Degree')
plt.legend((line1, line2),('Before attack','After attack'))
plt.show()
```

```
connectivity = nx.average_degree_connectivity(G, source='in+out', target='in+out', nodes=None, weight=None)
degree_list = list(connectivity.keys())
value_list = list(connectivity.values())
line1 = plt.bar(degree_list,value_list)
connectivity = nx.average_degree_connectivity(H, source='in+out', target='in+out', nodes=None, weight=None)
degree_list = list(connectivity.keys())
value_list = list(connectivity.values())
line2 = plt.bar(degree_list,value_list, alpha=0.75)
plt.title('Connectivity with random attack')
plt.ylabel('Connectivity')
plt.xlabel('Degree')
plt.legend((line1, line2),('Before attack','After attack'))
plt.show()

# Targeted attack
target_attack = do_target_attack(adj_matrix,5)
I = nx.Graph()
create_graph(target_attack, I, 0)

degree_dist = tab_degree_dist(adj_matrix)
line1 = plt.bar(degree_dist[0],degree_dist[1])
degree_dist = tab_degree_dist(target_attack)
line2 = plt.bar(degree_dist[0],degree_dist[1], alpha=0.75)
plt.title('Degree distribution with targeted attack')
plt.ylabel('Frequency')
plt.xlabel('Degree')
plt.legend((line1, line2),('Before attack','After attack'))
plt.show()

connectivity = nx.average_degree_connectivity(G, source='in+out', target='in+out', nodes=None, weight=None)
degree_list = list(connectivity.keys())
value_list = list(connectivity.values())
line1 = plt.bar(degree_list,value_list)
connectivity = nx.average_degree_connectivity(I, source='in+out', target='in+out', nodes=None, weight=None)
degree_list = list(connectivity.keys())
value_list = list(connectivity.values())
line2 = plt.bar(degree_list,value_list, alpha=0.75)
plt.title('Connectivity with targeted attack')
plt.ylabel('Connectivity')
plt.xlabel('Degree')
plt.legend((line1, line2),('Before attack','After attack'))
plt.show()
```