The Robustness Of Complex Networks

Daniel Catlin 110028893

23rd February 2018

Abstract

An introductory look at the robustness of several different networks models under both random and targeted damage using the size of the largest connected component and diameter of the networks as damage increases incrementally over time.

Introduction

Robustness

Put simply, the robustness of a complex network is its ability to withstand damage of different kinds and still behave as it usually would. This is an important feature of a network as complex networks can be used to model so many different things, their ability to still function properly under damage that often would occur in reality can give us insights as to how the network has its current underlying structure.

The two types of damage we will be looking at will be random damage and targeted damage. Firstly, by talking about a node that has been damaged, we mean the node has been removed from the network, and so any edges that had previously contacted the node have also been removed, for our purposes this will be done by removing the column and row corresponding to the node from the adjacency matrix.

For random damage, nodes will be chosen at random and removed from the network, all nodes that have not already been removed are equally likely to be chosen. For targeted damage the nodes will be removed in order of degree. The node with the highest degree being removed first, in the case where more than one node shares the highest degree then one of them is chosen uniformly at random from all with that degree.

Motivation

Complex networks can be used to model a huge amount of phenomena both natural and man-made, so knowing how these are affected by damage to the network could represent as many things as we model.

Random damage is very widely prevalent in large systems in which parts of the network my degrade over possibly through not being used or maintained or simply wear-and-tear. Such as in an electrical power network we may want to know what we could expect to happen to the network as a whole as pieces randomly break, and plan to structure the network using a model that is more resistant to random damage. Looking at a more natural example we could see the degradation of a human brain over time through perhaps aging or a specific neurological condition and determine what the effect would be on the patient as a whole.

Targeted damage could reflect the actions of malevolent actors in which they wish to cause maximum disruption to the system with minimum effort say as terrorism on a large transport network or pathogens attacking a protein interaction network.

Where the systems we are examining are not great in number but we have reasonably accurate network models that can replicate the structural properties, we can generate similar models and subject them to different types of damage in each scenario. As these networks are often very large, and repeated many times, numerical computation is the natural method to obtain results that can clearly show how we can expect a given model to perform when attacked in these ways.

Method And Functions

To show how the network is affected by the different types of damage, we will mainly be looking at the Giant Component of the network, and somewhat at the diameter also. The Giant Component being the largest connected component of the network, and the diameter being the largest shortest path between any two nodes in the network.

As a method of notation, when referring to a function or a script called we will be using the convention that it will be given in italics and where objects are assigned names, this will be in quotation marks.

The script *networkgeneration* is used to generate the networks that are examined and is included with the code, this is to allow networks to be created and so further scripts can damage them and generate results using the same network each time without having to regenerate the network each time. Although it must be noted that the networks will vary slightly each time the script is run for each user, so the specific results may vary, but during the experimentation the results were largely the same as the networks were large enough to allow small differences so should not invalidate the results.

Firstly, we will determine whether we expect the network to have a Giant Component, this will be done based on the Molloy–Reed criterion shown in the inequality below:

$$\frac{<k^2>}{<k>} > 2$$

Where $<k>$ and $<k^2>$ represent the first and second moments of the average node degrees and if this inequality is true, the we can expect there to be a Giant Component. The script *GCcheckervalues* will calculate this for the larger networks that are generated by the *networkgeneration* script.

From this we will calculate the critical threshold f_c which is the fraction of nodes of the network we would expect to have to be removed to disrupt the Giant Component in to no longer having a size of the same order of magnitude as the total number of nodes in the network. Calculated using the following equation:

$$f_c = 1 - \frac{1}{\frac{<k^2>}{<k>} - 1}$$

To do this, I have created two functions named *GCchecker(adjmat)* and *criticalfraction(adjmat)* respectively, to be found in the attached code, both of these taking a (sparse) adjacency matrix as their only argument, the code is commented as to show what is happening in the script at each stage. The script *criticalfractions* is used to calculate these values for the networks generated by the *networkgeneration* script.

Secondly, once we have obtained these, we will use the created function *GCsize(adjmat)* to return the size of the largest connected component of the network, which takes as its only argument an adjacency matrix, once again for clarity, the code contains all explanation of the workings of it commented inside as this function is broken down in to sub-functions to find different components. These being *compfind(adjmat, components, node, compnum)* and *compfindall(adjmat)* in which the second calls the first, so only the adjacency matrix is needed, and this is called by the previously mentioned *GCsize(adjmat)*, further explanation included in code for clarity.

To show the network undergoing damage we will record the size of the largest connected component with *GCsize(adjmat)* and the remove a number of nodes. This is done using the created function *randomdamage(adjmat,nodes)*, which takes for its arguments an adjacency matrix and a number of nodes to be removed, returning the new adjacency matrix after the damage. This process is repeated until all nodes have been removed and we use the results to produce a graph of the size of the Giant Component against the number of nodes that have been removed, and this will be compared to our theoretical value of the critical fraction.

This process will then be repeated for targeted damage in the same way, except using the function *targetdamage(adjmat,nodes)* included rather than *randomdamage(adjmat,nodes)* to produce a similar plot.

Finally, the above procedure will be used again for the networks under both types of damage with the difference being we will calculate and record the Diameter of the network as opposed to the size of the largest connected component using the created function *diameter(adjmat)* which takes the adjacency matrix as its only argument. Although this will be done on a network of a smaller size as the function requires $O(NK)$ operations, where N is the number of nodes and K is the number of edges of the network and would be unfeasible on networks of the size used in the first part.

1 Erdős Rényi Model B

Firstly, we will be looking at the Erdős Rényi Model B, networks of which can be generated with the function *ERmodB(N,p)* included in the code which generates a network with N nodes and probability p for the creation of a link between any two nodes. The algorithm for this is simply to create the number of nodes required and taking every potential pair, sample a random variable for the uniform interval [0,1] and create an edge if this random variable is less than that of the value of p, giving edges created with this likely hood. Done by looping over the rows inside a loop over each column, more clearly seen through the comments in the code provided.

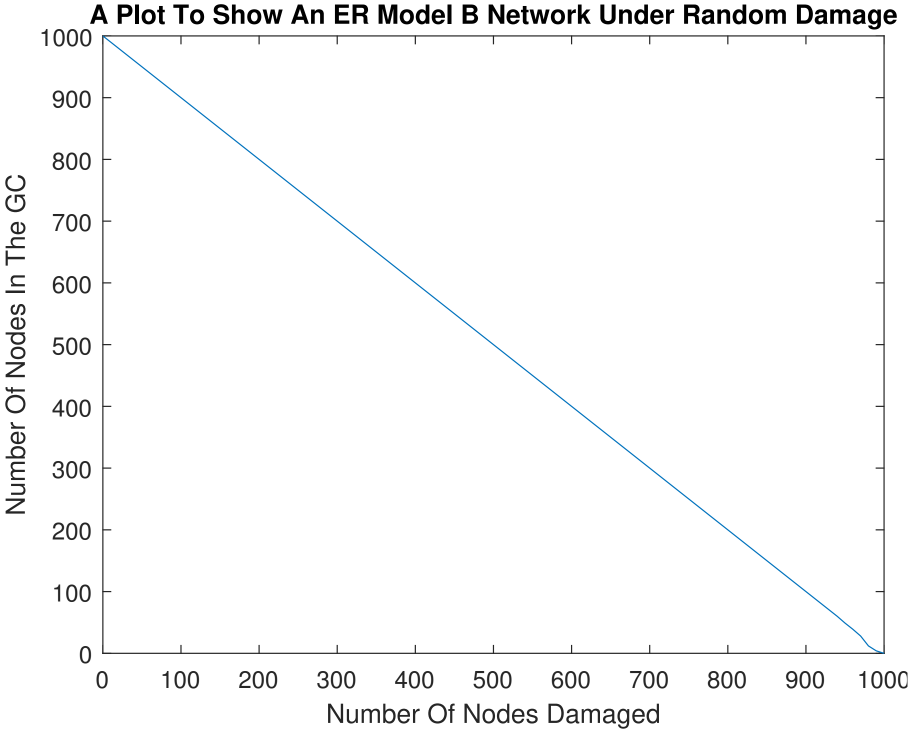
The script *networkgeneration* creates two networks using the function calls *ERmodB(1000,0.1)* and *ERmodB(100,0.1)* and assigning them names 'ERmodBnetwork' and 'ERmodBnetworksmall' respectively. Below we see the equation for the expected number of edges K such networks would have (not to be confused with the value K in the Erdős Rényi Model A which is fixed, here it is variable as p is fixed).

$$K = \binom{N}{2} p$$

The networks generated for this in-fact had expected values of this as 49950 and 495 for the larger and smaller networks respectively. The ones used in testing had actual values of 49781 and 476 also respectively, in line with what we would like to see.

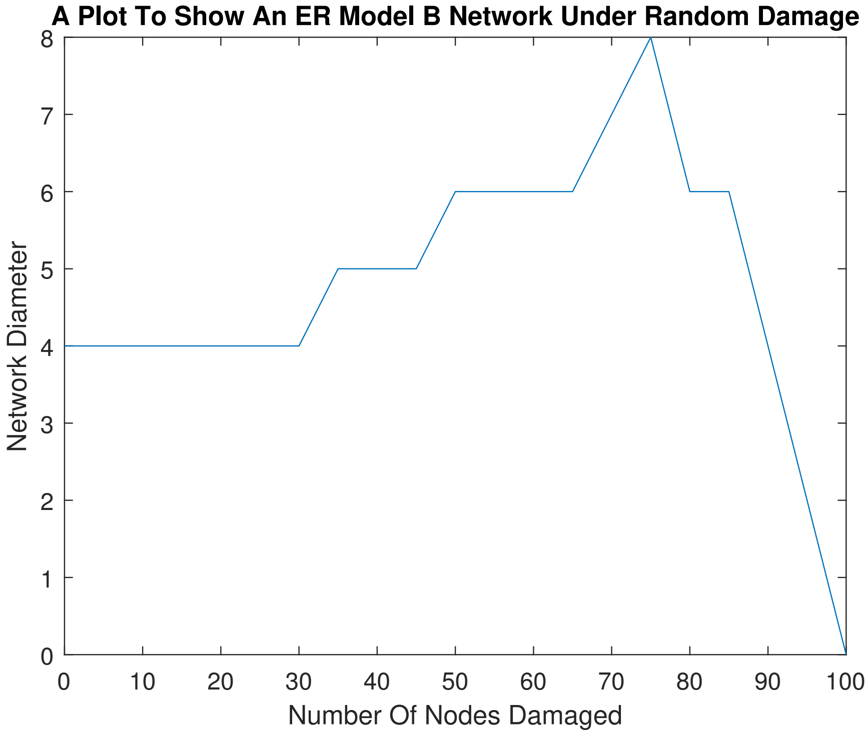
1.1 Random Damage

Using the script *GCcheckervalues* we see that this network returns a value of 100.56 so we can expect the largest connected component to contain most if not all of the nodes. Using *criticalfractions* we see a value of 0.99 is given, showing that we would need to remove 99% of the nodes before the network breaks in to a few disconnected components, we now run the script *ERmodBrandomgraph* to subject this network to to random damage until we have removed all of the nodes.



So we can clearly see from this, the linear decrease of the largest component as the nodes are removed. As the nodes would all have a roughly similar degree and none would be any more essential to the network than any other on this scale this would make sense, as there is not really any node that is more important than others that can randomly be removed. We see also in-fact that initially the network was one connected component and we did have to remove nearly all of the nodes for the largest component to be less than the total number of nodes in the network as seen by the slight dip at the bottom of the graph.

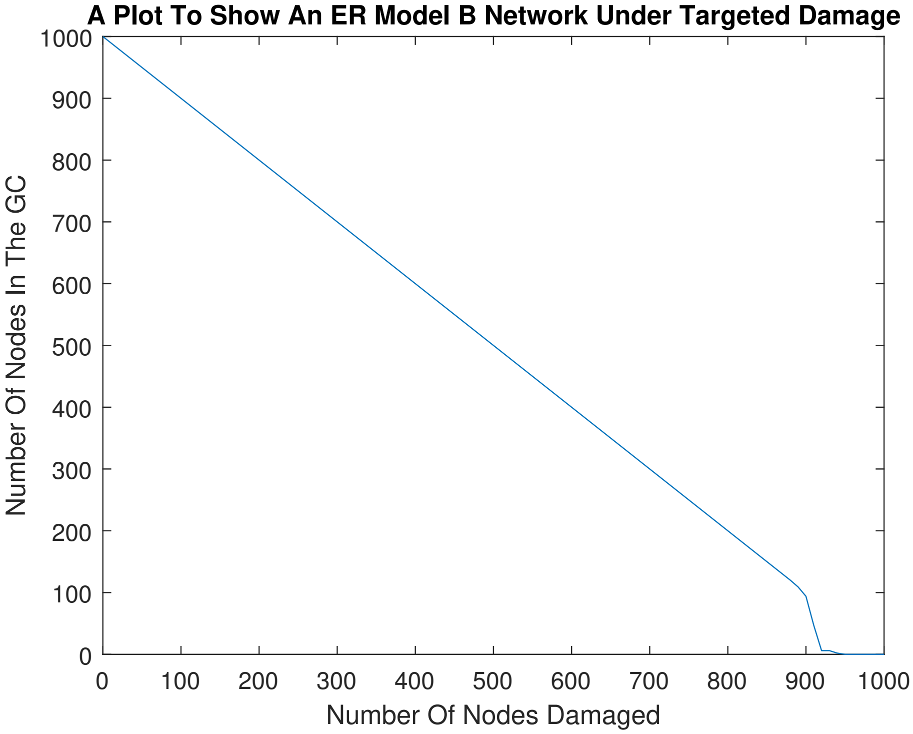
We now would like to look at the network while being randomly damaged but instead looking at the diameter after each successive iteration of damage. Initially this was run on the same network but due to the amount of operations required this was still running after 4-5 hours so instead a smaller network was made in which this could be done in a more timely manner. The *networkgeneration* script creates a network of N = 100 nodes using p = 0.1 so we would expect the average degree to be smaller in this case. The script *ERmodBrandomdiametergraph* is then used and we obtain the following graph.



So we now see the diameter increase incrementally as nodes are removed and the network, although of a smaller size, is harder to traverse. This increases to the point where the network contains only 20 nodes and has a diameter of 8, showing it may have had a structure much closer to that of a tree compared to the initial structure. Although we see the diameter decrease and drop to nothing as the network becomes more disconnected and contains few nodes of which paths can exist between.

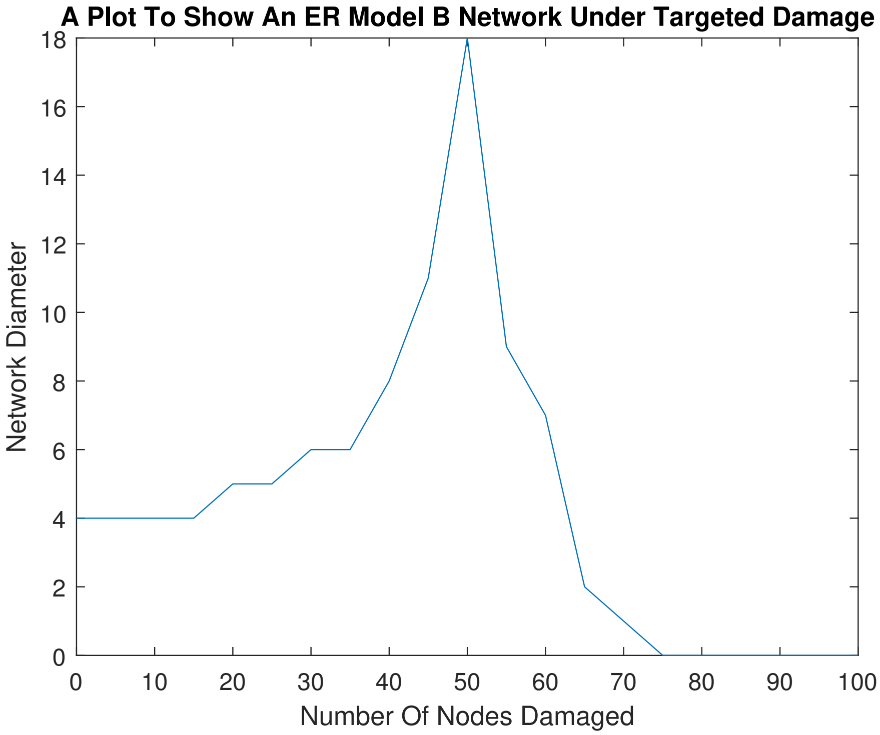
1.2 Targeted Damage

Now applying the script *ERmodBtargetgraph* to remove nodes of the highest degree first to this same network we would imagine the results to be very similar to that of the random damage as the nodes in general all have a similar degree, although approaching the end, we could expect the largest connected component to be slightly more fragile as we are left with the lowest degree nodes that are unable to connect much to each other.



Indeed we see that after 90% of the nodes are removed, we have a sudden drop in the size of the largest connected component which shows some improvement over the 99% needed with random damage. As this network is not really expected to contain any large hubs this would agree with what we would expect in that the a network that is largely homogeneous is similarly resistant to damage this is random or targeted as the largest nodes for the majority of the damaging process.

Now we proceed to run the script *ERmodBtargetdiametergraph* on the smaller network of this model we have and see how the diameter is affected by the removal of nodes of the highest degree first.



Here we achieve what may be the most interesting result for this model, as removing the highest degree nodes drastically increases the network diameter from 5 to 18 when between 20 and 50 nodes are removed. This may indicate that once some nodes are removed initially (between 0 and 20) we see a few more that although have a similar degree to the previous, would presumably have had a higher betweenness and such after their removal made the networks shortest paths much more *round-about*.

This continues until we only have half the network left and then we see that it has become broken down in to many small connected components and as such, the diameter drops from 18 to 0 after between 50 and 75 nodes are removed. Beyond this point we also note that the network is entirely disconnected components as the diameter is 0 and no paths even exist anymore.

2 Barabási Albert Model

Secondly we have the Barabási Albert Model in which as we add nodes they will be more likely to form edges with nodes of a higher degree. We see this captured in the following equation:

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

Where p_i is the probability of a new node forming an edge with an existing node i, k_i is the degree of node i and the sum of k_j is the sum of the degrees of all nodes in the network already.

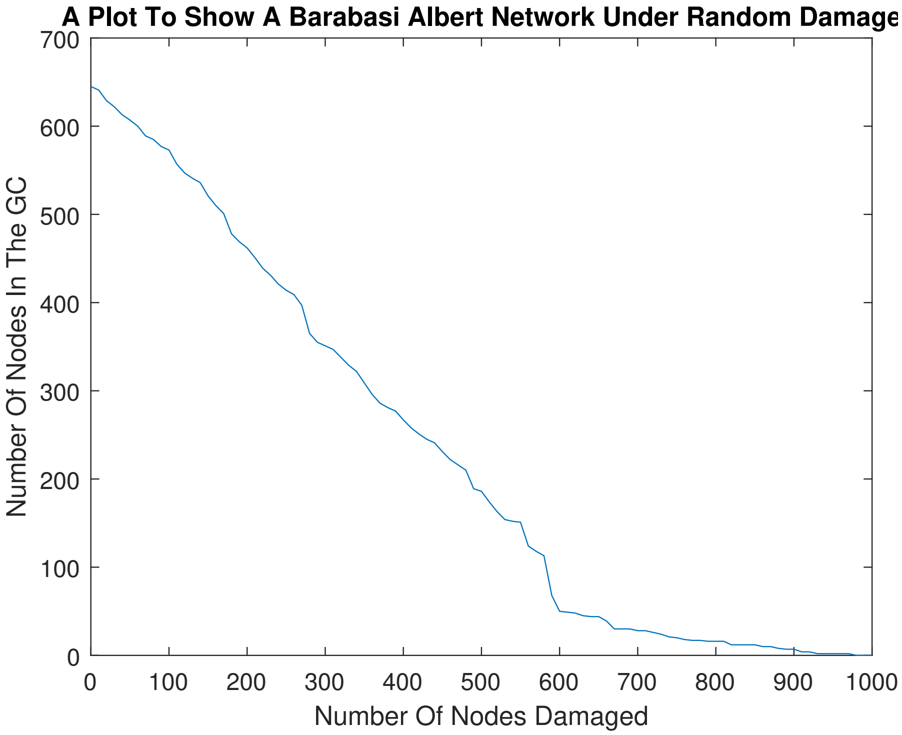
This method of preferential attachment allow the networks we generate to be scale free in nature and as such means a power law degree degree distribution so we see a very few nodes with a degree much higher than the average for the rest of the network, these nodes are often called its hubs and allow some more interesting properties to be seen such as a greater resistance to random damage we will hopefully see.

As seen in the code attached, the algorithm for generating these networks starts by taking a small seed graph of which to start from and then at each time step adds a node and creates a link with a current node with probability seen above. This is then repeated for each node that exists and then we consider the new node part of the network and repeat the process for the addition of the next.

With respect to the function used to generate this, it is called with *BAmod(N,seed)* to generate a network with a total number of nodes N and a given number of starting nodes as the seed parameter. To give a fair starting point, the number of seed nodes are linked in a single cycle and as such, must be a value of at least 3 with the minimum starting point being a triangle graph. For the graphs we are looking at, the *networkgeneration* script creates two networks of this type using *BAmod(1000,3)* and *BAmod(100,3)* assigned 'BAnetwork' and 'BAnetworksmall' respectively.

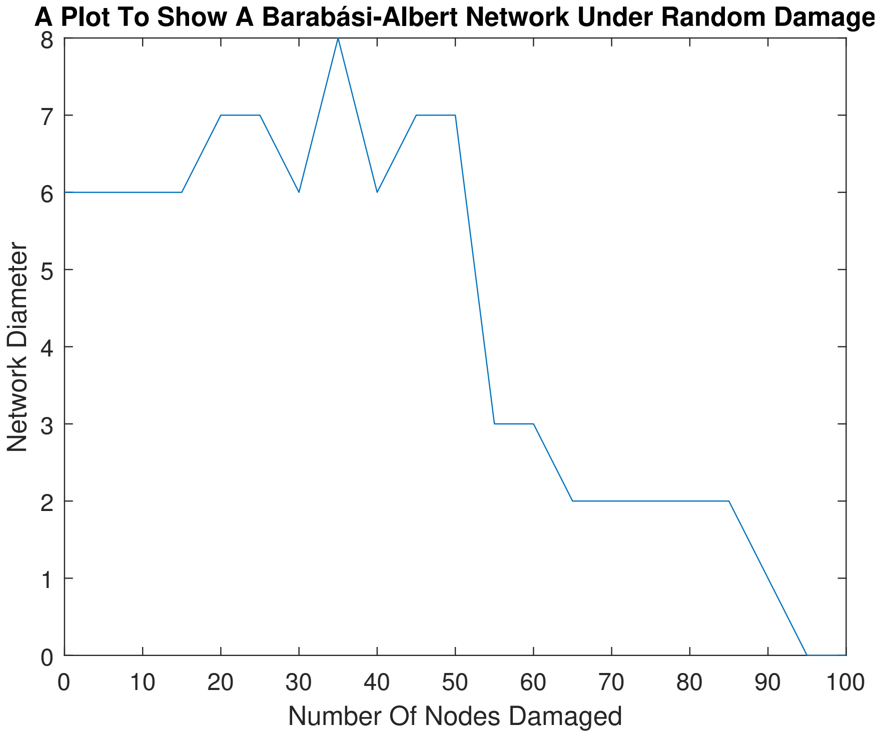
2.1 Random Damage

As similar to before, we have a script *BAmodrandomgraph* to calculate the size of the largest connected component and incrementally damage the network by the removal of nodes at random. Before this though, we see a value of 9.39 returned by the *GCcheckervalues* script for this network indicating the existence of a Giant component and a critical fraction from *criticalfractions* of 0.88 indicating we may need to remove up to 88% of the nodes before we see it disappear.



Here we see size of the largest connected component decrease almost linearly like the Erdős Rényi Model B network for most of the process although we can clearly see the point at around the removal of 600 nodes that this component drops off much quicker, presumably due to the loss of the last few remaining hubs that have avoided be taken before had due to the large amount of lower degree nodes. This is also somewhat earlier than we would expect from the critical fraction value as it was around 60% node removal in which this happened. This does show that when subjected to random damage this model is quite resistant up to a point and although the previous model did fare slightly better, it contained a lot of unnecessary links and as such in a real world example such as building a power distribution network this model would be much more economical and give adequate resistance to its own random failures.

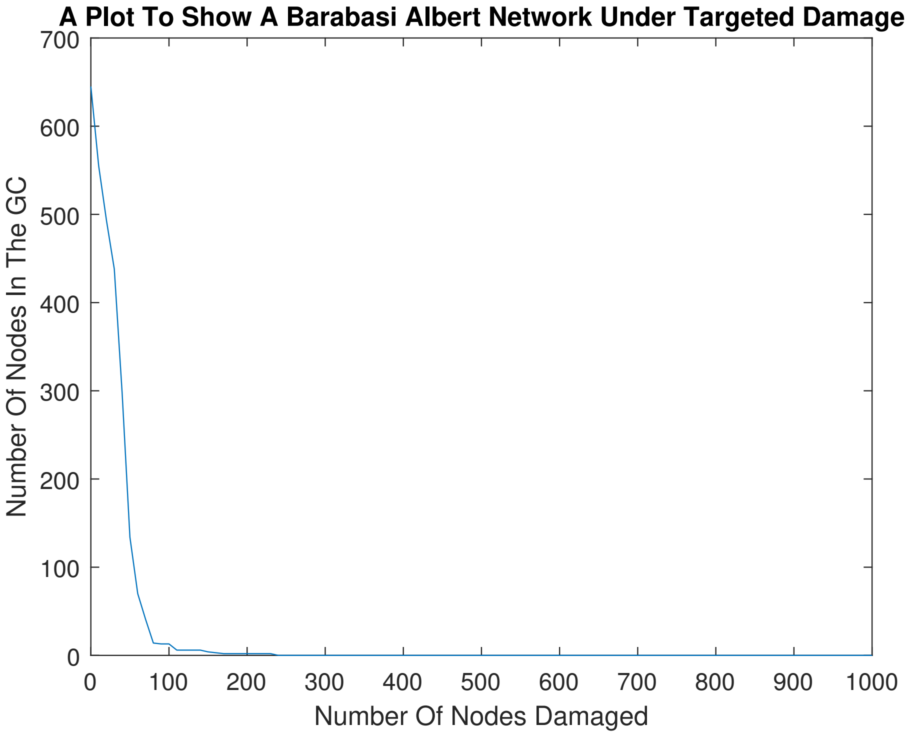
For this next result, we use the script *BAmodrandomdiametergraph* to repeat the previous but on the smaller of the two networks, calculating the network diameter each time instead.



Now here we see that when nodes are removed at random, there is a fluctuation of the diameter but as the nodes that are removed are mostly the much more common low degree nodes we don't really see much of a change other than movement between a diameter of 6 and 8 up until we still have at least 50 nodes present. Although beyond this we see that once we have more than 50 nodes deactivated we see the diameter start to decrease rapidly as the network is smaller and broken in to more parts which contain much smaller paths, possibly due to the loss of some of the hub nodes holding different parts of the network together being lost.

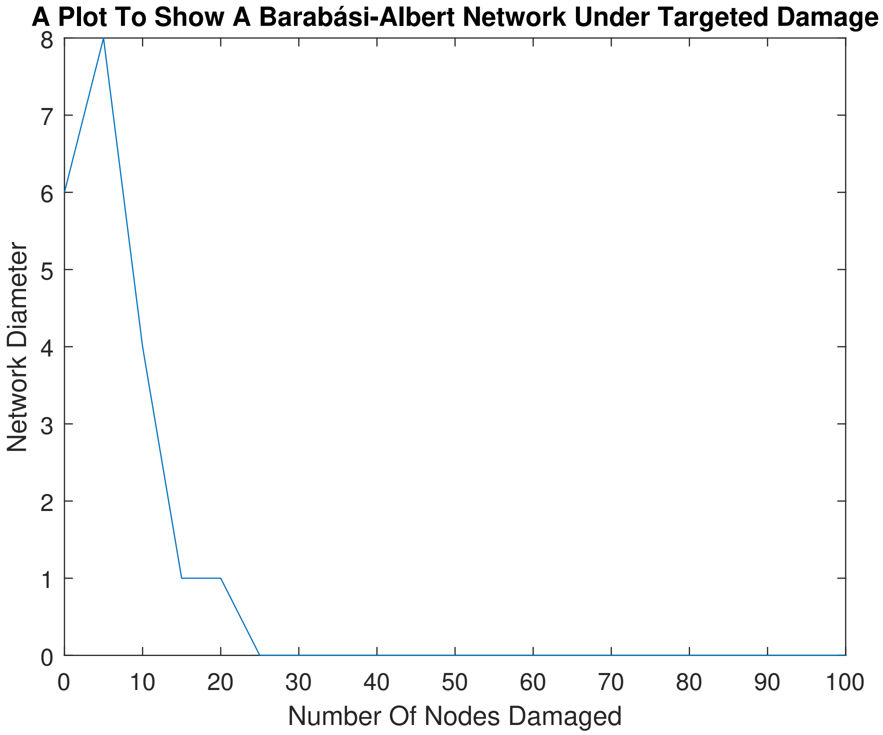
2.2 Targeted Damage

Now, under the removal of nodes with the highest degree first we would expect to see the giant component disappear very quickly as the hub nodes are taken first, leaving the lower degree nodes connecting smaller and smaller disconnected components. Similarly, the process is called using the script *BAmodtargetgraph* and returns the graph below for this, using the larger network of 1000 nodes.



As we would expect, the graph shows that the size of the largest component which initially holds around 65% of the nodes, is decimated after the removal of the top 10% and beyond the removal of more than 100 nodes, we can assume the network is simply a large collection of much smaller connected components. Hence we can very clearly see the expected property this model has in which its Giant Component is comparatively resistant to random failure but very weak to targeted attacks.

Next we subject the smaller network of this model to rounds of targeted damage, examining the network diameter, this graph being generated using the included script *BAmodtargetdiametergraph* as follows.



For the removal of the first 10% of the nodes, we see the diameter increase from 6 to 8 as the hub nodes are taken which link opposing areas quickly but still are connected. After this, we see the diameter fall to 1 and then 0 as 15 and 25 nodes are removed respectively. We would understand this is as further hubs are taken, the network becomes a series of small disconnected components in which there are pairs of nodes giving a diameter of 1 and then disconnected nodes giving a diameter of 0. Once again this shows the fragility of this network model to to targeted damage compared to the similar results under random damage in which we have to remove around half of the nodes for this same effect to occur.

3 Bianconi Barabási Model

The final model we will be looking at is the Bianconi Barabási Model, or Fitness model. This is largely similar to the previous model, with the main difference being that the probability of an edge being created with an existing node not only relies its share of the links but also its fitness in relation to that of the other nodes as well. Each node, when created is given a fitness value taken from as separate distribution. This allows the network to grow and have hub nodes as before but allows nodes added later to potentially overtake older nodes and change the structure somewhat during growth, which is not present in the Barabási Albert Model as older nodes will always be more likely to hold a higher number of links

The algorithm to generate this network is very similar to that of the Barabási Albert Model in which a starting amount of seed nodes are given and at each time step a new node is added and connected to the existing nodes, although in this case, when a new node is created, it is given a fitness η taken from the distribution $\rho(\eta)$ with the probability p_i that a new node is connected to an existing node i follows the equation:

$$p_i = \frac{\eta_i k_i}{\sum_j \eta_j k_j}$$

Where η_i and k_i represent the fitness and degree of node i respectively, and the sum of the product of the fitness and degree of all existing j nodes is shown by $\sum_j \eta_j k_j$.

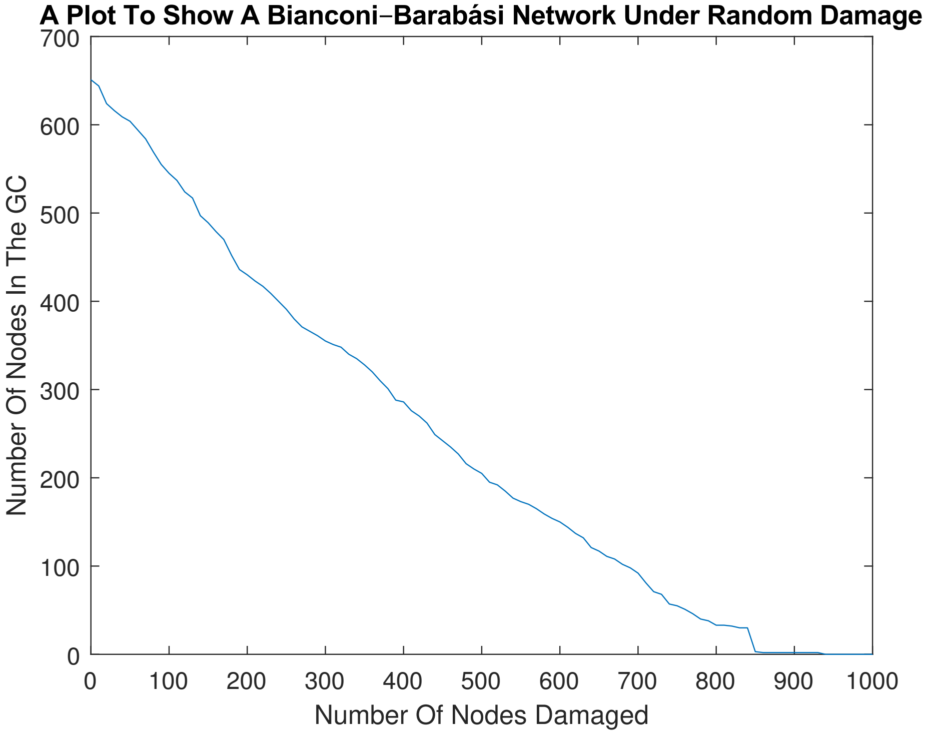
As to the code used to generate these networks, we have a created function *BBmod(N,seed)* which has for its arguments N for the total number of nodes in the resultant network and seed as the initial number of nodes to grow the network from. As before, the value of seed will start the network from a graph that is a single cycle of this length and as such must be at least 3 and allows for an unbiased starting point. Furthermore, for the fitness distribution $\rho(\eta)$ we use a uniform distribution over [0,1], noting that although this does allow a node to be assigned a fitness of 0 and so unable to attract any edges, this is very unlikely and could still be a behaviour seen in the system being modelled.

The script *networkgeneration* as seen previously, will generate 2 networks of different sizes using the commands *BBmod(1000,3)* and *BBmod(100,3)* and assigns them 'BBnetwork' and 'BBnetworksmall' respectively. As seen in the code, this recalculates the probability for an edge to be added every time which is then looped over for all existing nodes and as such, is very time-consuming.

Based on testing with MATLAB's tic/toc functions, generating a network of 1000 nodes using this takes around 18 minutes on my 6 core PC, much longer than the previous 2 models which took less than 1 minute each. So if further research in to this was needed, I would seriously consider restructuring the code to be more efficient in its generation.

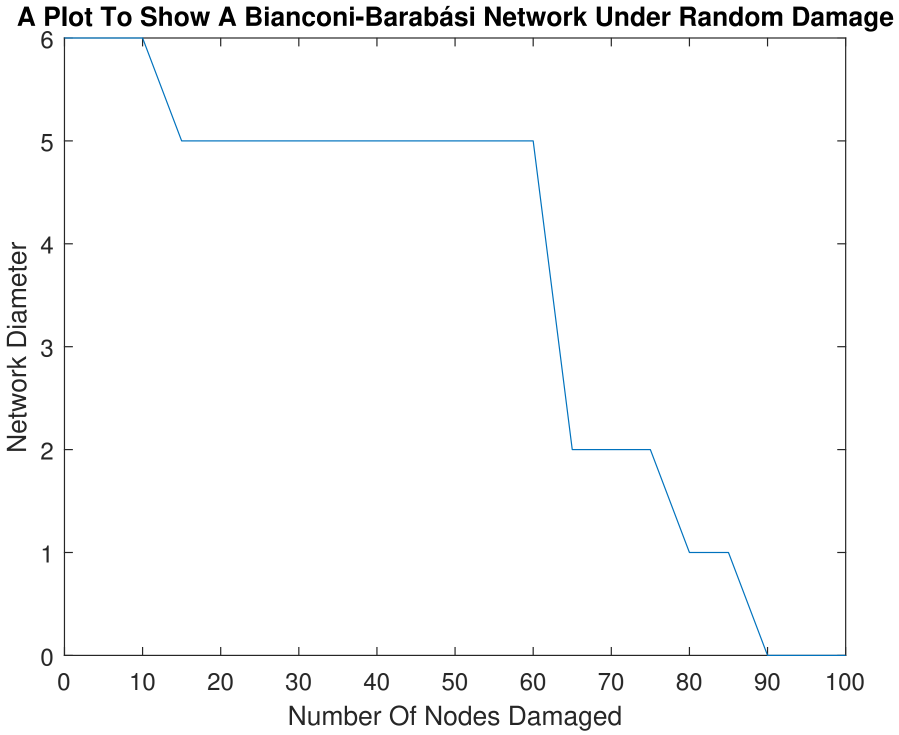
3.1 Random Damage

As this model is quite similar to the Barabási Albert Model we would expect it to perform similarly under random damage as that did as we still have a few hub nodes among many others. Using the scripts *GCcheckervalues* and *criticalfractions* we are returned values of 28.61 showing we can expect a Giant Component and 0.96 meaning we would need the removal of 96% of the nodes to disrupt this under this damage regime. We now call the script *BBmodrandomgraph* to achieve the following.



As we saw in the previous model under similar damage, we have the largest connected components size decreasing almost linearly over time, disappearing after around 85% of the nodes are removed, slightly earlier than with the expected 96% given as the critical fraction. Although we see that this model was more resistant to the damage than the previous as it needed around 85% damage versus 60% of nodes removed in the Barabási-Albert Model. This could potentially be attributed to the fact that the nodes of a much higher degree are more evenly distributed through the network than before so the loss of each, although unlikely, has a smaller effect on the size of the Giant Component as the links are spread around more due to fitter nodes added later taking a greater share than nodes added later in the previous model that are unable to achieve such a degree.

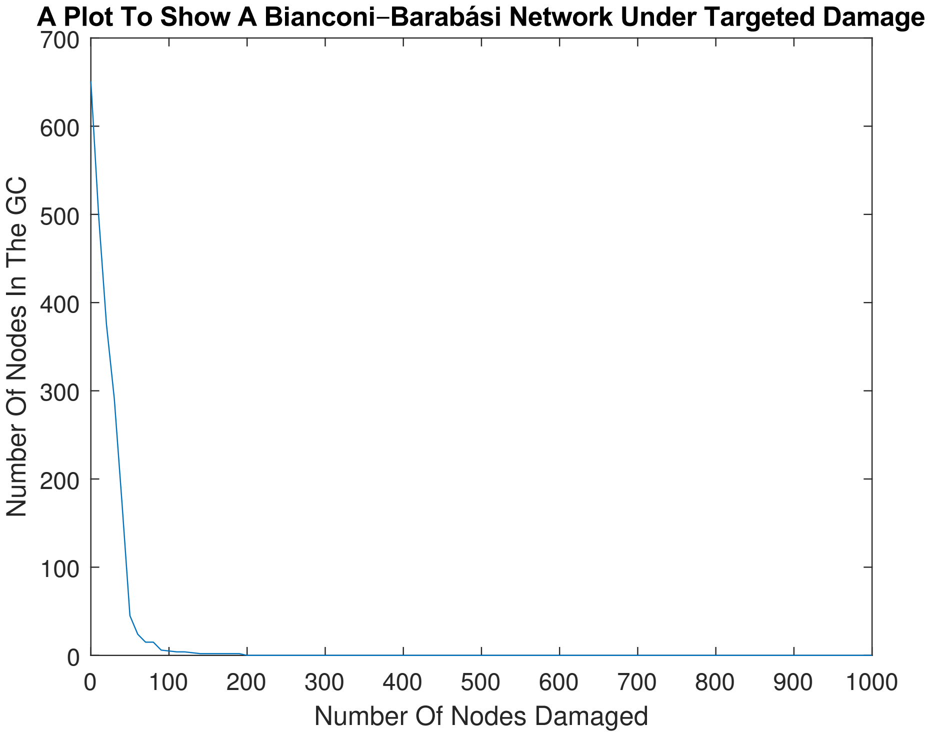
As before, we use the script *BBmodrandomdiametergraph* to remove nodes randomly from the smaller graph, looking at the networks diameter each time and produce the graph below.



Interestingly the networks diameter is only decreased by 1 when 15 nodes are removed and maintains a value of 5 until more than 60% of the nodes are removed. We would attribute this to the majority of nodes being lost before this point being lower degree nodes that did not contribute much to the diameter, until we suddenly lose a few much more important nodes which then causes the network to breakdown in to separate disconnected components in which the diameter is only 2, then decreasing over the loss of the next 30 nodes to 0. We can see this is very similar to the result of the previous model under the same conditions, minus the fluctuations at the start, but both need around 60% of the nodes removed before the network breaks down.

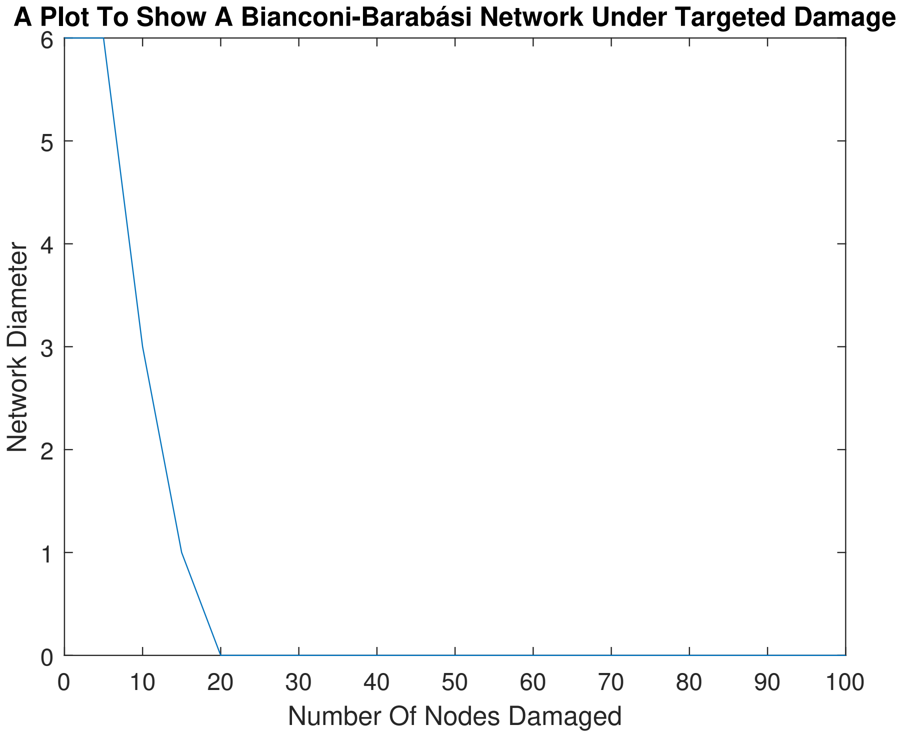
3.2 Targeted Damage

Finally we move on the the last examples of targeted damage and once again we could expect similar results for this model as we saw with the Barabási-Albert Model in which the networks are much more fragile to this form of attack. As before we call the script *BBmodtargetgraph* which acts on the larger graph.



As we would have expected, under targeted damage the Giant Component is almost immediately removed and reduced from a size of approximately 650 to around 10-20 before we have 10% of the nodes removed. This is very similar to that of the Barabási-Albert Model under this attack, and shows that as the important hub nodes make up less than 10% of the network, the difference in their distribution does not really affect this outcome and the network collapses as before, at a similar amount of damage.

Lastly, we call the script *BBmodtargetdiametergraph* to see the effect this attack has on the diameter of the smaller network., potentially expecting it to be similar to that of the Barabási-Albert Model.



Here we do indeed see a result very similar to before in that after the removal of only 20% of the nodes, we have completely reduced the network to disconnected nodes and get a value of 0 for the diameter after this point. This is at a similar point to before and what was expected as key nodes that connect different parts are removed first and create many disconnected components, in line with how it happened in the last model.

Conclusion

As we have seen in the examples discussed for the models examined, for the most part the predictions made were similar to the results obtained although examples such as the critical fractions were higher than what was seen in reality. This difference could be a further area of interest if further work on this was carried out. Further to this, a more efficient way to calculate the network diameter would be useful as it would allow the analysis to be carried out on networks all of the same size and we would imagine, produce stronger results.

In a similar manner, more efficient functions to create the networks would be useful, predominantly for the Bianconi-Barabási Model which was cumbersome in its execution. A further model of interest that could also be looked at in this way would be the Configuration model even if just to confirm the assumptions of how it would work. Further to this, the resistance to random and targeted damage on multilayer networks in which the layers are of the same or even different types would be interesting to see and potentially could be done with largely the same code, albeit with some modification for generation and the measure we are taking.

The code included does also contain a function named *ERmodA(N,K)* which creates an Erdős Rényi Model A network with N nodes and K edges but was eventually omitted as deemed too similar to model B to warrant further discussion, but did provide a good starting point for the code structure of the models used.

Lastly, it should be mentioned that attempts were made to include a section testing the robustness of the neural network of the C.Elegans worm to see how a real world example looks, this network data was obtained from http://www.complex-networks.net/datasets.html and included with the code. I acknowledge this was downloaded and not my own work but has been included for completeness. Unfortunately a solution to be imported in to MATLAB for analysis was not found but my prediction would be that as something that has evolved over a long period of time would be reasonably resistant to random damage which would mimic neurons failing over time but would be much more susceptible to a targeted attack as these are much harder to defend against but also much less likely so not as important to expend energy in creating defenses for.