Kyle Hagood, Syed Rizvi

Social Networking and Analysis

CMSC 412-001 Final Report

Analyzing the Resiliency of Strongly Connected Networks in Combatting Recursive Destruction

Highly connected networks, sometimes referred to as “small-world networks” are an incredibly important part of everyone’s lives, especially so in the modern age. From the neural networks in our brains that allow us to process information, to the road maps we use everyday to transit, to the power grids that keep our electrical devices functioning, without these systems in place and functioning properly, we could be swept right back into the stone age… or dead. It was with this realization that our group began to ponder over just how resilient these networks truly are. And over how we could even test such networks to determine their resiliency.

In a naive approach, we originally conceded to employ a scheme where a computer program would recursively delete the most important nodes based on betweenness centrality. We would then calculate how many iterations of deletions it took before at least one node in the graph was disconnected from the rest. What we learned after only a day or so after, however, was that there already existed an algorithm that did just this with a mathematical formula. So, that idea had to be scrapped. At this point, we knew that we needed to search for a more novel approach to the game we were constructing. And here, we realized another flaw in our original plan. In explanation, another glimpse into the naivety present in our first attempt is provided by the realization that betweenness centrality in real-world models doesn’t necessarily correlate strongly with the importance of that node. Certainly in some cases it would, but not all.

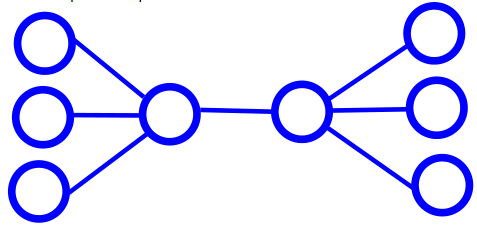
For a moment, let’s consider the example of a network of power grids. In this type of graph, every node would represent a power line pole, and every edge would represent an actual power line connecting two power line poles. It is reasonable in this type of example to infer that betweenness centrality would often be a good measure of the level of importance for a node. Since electricity is distributed equally along each line, taking away the power line pole with the greatest betweenness centralities means there is a greater chance of sections of the overall power grid becoming disconnected, and therefore losing power within that entire disconnected component.

Now, consider for a moment the example of a graph in which each node represents an agency of some sort, and each edge represents a line of communication between those agencies. An agency could be anything here - a post office, the NSA, a secret spy organization, anything. And let’s consider the idea that some agencies within the overall graph hold incredibly important information which, in its absence, would cause the rest of the agencies within the graph to fail. In this kind of situation, it is reasonable to infer that such agencies would want to be kept secret, and might be aware that a vast number of communication lines directed towards them could compromise that agency’s status as a holder of important information. In this sort of representation, betweenness centrality would not be a good measure of a node’s importance even in the least bit. The most important agencies might actually be more inclined to have small betweenness centrality, opting instead to have few very secure and private communication channels so as to avert an abundance of attention, or suspicion.

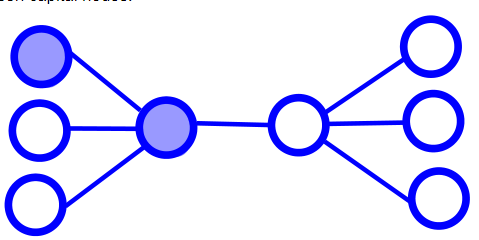
We needed an approach that was both novel as well as generic, an approach that is creative in its operation, and can be applied to a diverse array of graphs in order to determine the resiliency of that graph in a situation where any number of nodes could, in fact, be the most important nodes. We needed a game.

The Game

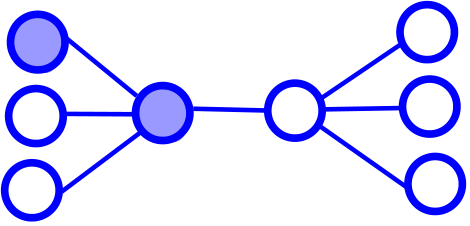
In order to visualize our algorithm, first imagine a graph. It could be any graph but here we will use a simple example.

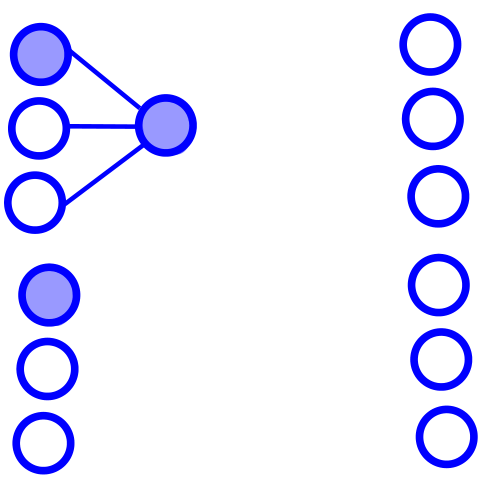


Now imagine that, in line with our previous realizations, any node in this graph has an equal chance of existing as a highly-important node. Let’s call these highly-important nodes “capital nodes”, and use a random number generator to mark them.



Finally, imagine there exists an adversary with two extraordinary powers. First, the power to magically calculate the coordinates of the most important node in the graph, based primarily on betweenness centrality. And second, the power to magically and iteratively obliterate these most important nodes from existence. Remember here that a most important node isn’t usually a capital node, although it certainly can be. Let’s assume that the adversary’s goal is to disconnect all lines of information coming to or from at least one of the randomly chosen capital nodes. As can be seen below, this is how that goal would be accomplished.





The Algorithm

Now that we’ve gone over how the game works, we will discuss the algorithm used to execute this game. Seeing as we developed what is to the extent of our knowledge a novel approach, there was no way around piecing together a custom algorithm to achieve our goals. One major and pre-existing piece to the puzzle, however, was the ability to create graph objects and calculate node’s betweenness centralities. After a bit of searching, we decided to use the library created by “Anastasia Kurdia” [1], and developed the remaining portions of the algorithm by hand.

The Pseudocode

*public void IterativeDestruction(adjacencyMatrix){*

*buildGraph(adjacencyMatrix); [1]*

*betweennessArray[] = calculateBetweennessCentrality(adjacencyMatrix);*

*degreeArray[] = calculateDegreeCentrality(adjacencyMatrix);*

*capitalNodes[] = randomlyChooseCapitalNodes(adjacencyMatrix);*

*boolean done = false;*

*while (!done){*

*node largest = pickLargestBetweennessCentralityNode(betweennessArray);*

*if (largest == 0){*

*Largest = pickLargestDegreeCentralityNode(degreeArray);*

*}*

*deleteLargestFromGraph(largest);*

*if (anyCapitalNodeDisconnected(adjacencyMatrix, capitalNodes))*

*done = true;*

*else{*

*buildGraph(adjacencyMatrix); [1]*

*betweennessArray[] = calculateBetweennessCentrality(adjacencyMatrix);*

*degreeArray[] = calculateDegreeCentrality(adjacencyMatrix);*

*}*

*outputIterationInformation(new fileName);*

*}*

*}*

In the above pseudocode, every occurrence of the [1] annotation references the use of the library provided by “Anastasia Kurdia”. The “new fileName” field represents a fileName that would be passed in through a command line argument, so that one can avoid overwriting previous output files with the new.

Further, it can be seen that we check for the case in which the greatest betweenness centrality calculated was found to be zero. We discovered, through testing, that in this case our algorithm would enter an infinite loop. This was amended by adding the line to alternatively use the largest degree centrality, which would never be zero unless the algorithm is finished executing anyways. The actual implementation of the algorithm also creates a new directory for each test that is run, so that on each destructive iteration a new graph file and relevant data file is constructed. This will allow us to analyze the data, if necessary, even iteration by iteration.

Moving Forward

So far, in addition to implementing the algorithm, which was most likely the most rigorous challenge in the scope of the project, we’ve begun testing on two different graphs. These are the neural network as well as the power grid network, both compiled by “D. J. Watts and S. H. Strogatz” [2]. We chose to temporarily convert the neural network from a directed graph to an undirected graph at the current time. On each of the graphs, we have collected the raw data from five different testing iterations, where one testing iteration is considered complete with one full execution of the aforementioned pseudo code algorithm.

Polling from this raw data, we can view some unprocessed information as follows:

#### Neural Network 297 Nodes 2345 Edges

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test 1 | Test 2 | Test 3 | Test 4 | Test 5 |
| 3 Iterations | 62 Iterations | 7 Iterations | 17 Iterations | 6 Iterations |

#### Power Grid network 4941 Nodes 6594 Edges

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test 1 | Test 2 | Test 3 | Test 4 | Test 5 |
| 135 Iterations | 489 Iterations | 603 Iterations | 207 Iterations | 518 Iterations |

Using the power of Gephi, and through developing an algorithm to output .csv files of the new adjacency matrix after each iteration, we are also able to visualize the before and after scenarios for each graph. To reiterate, the after scenario is when there exists a randomly selected capital node that is disconnected from the rest of the graph.

#### Neural Network 297 Nodes 2345 Edges

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Original | Test 1 | Test 2 | Test 3 | Test 4 | Test 5 |
| NeuralNetworkUndirected.csv-Original.png | NeuralNetworkUndirected.csv-test1.png | NeuralNetworkUndirected.csv-test2.png | NeuralNetworkUndirected.csv-test3.png | NeuralNetworkUndirected.csv-test4.png | NeuralNetworkUndirected.csv-test5.png |

#### Power Grid network 4941 Nodes 6594 Edges

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Original | Test 1 | Test 2 | Test 3 | Test 4 | Test 5 |
| Power.csv-Original.png | Power.csv-test1.png | Power.csv-test2.png | Power.csv-test3.png | Power.csv-test4.png | Power.csv-test5.png |

In our next and final steps in the project, we plan to continue collecting raw data for various graphs. Once this is complete, which shouldn’t take long, we will need to devote our time to brainstorming the best way both graphically and mathematically to present our data once it is analyzed. What we hope to find is some interesting pattern or trend in the data that could suggest where weak spots in a network are likely to exist in relation to this destructive randomized scheme. Once we locate some pattern or trend, we will likely be taking steps to explain in each mathematical, graphical, and human terms the suggested best practices for eliminating these previously mentioned, hypothetical weak spots in a network.

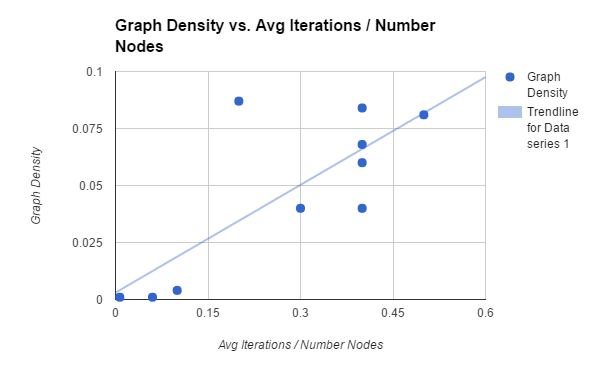
In our final project iteration, we continued the collection of undirected graphs until we were up to ten in total, then ran them through our iteratively destructive algorithm for a total of five tests each. For good results, we ensured that the graphs used had a wide range of node sizes, edge sizes, and densities. After the tests were run, we compiled the information into the tables below.

The leftmost graph is a simple self-explanatory compilation of data. The rightmost graph shows each of the Gephi visualizations for each graph in its original version, then after each of the test runs. The graph below these two is one that shows our data’s relationship with respect to the average number of iterations it took to ‘destruct’ the graph according to our definition, as well as the density of the graph. We have normalized the average number of iterations to destruction through dividing by the number of nodes in the graph.



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Nodes | Edges | Graph  Density | Avg  Iterations |
| Airlines | 235 | 2594 | 0.047 | 82.4 |
| Adjnoun | 112 | 425 | 0.068 | 45.0 |
| Celegans | 306 | 2345 | 0.046 | 139.4 |
| Codeminer | 724 | 1025 | 0.004 | 93.0 |
| CpanDist | 2724 | 5018 | 0.001 | 168.4 |
| Dolphins | 62 | 318 | 0.084 | 27.4 |
| Lesmis | 77 | 254 | 0.087 | 22.6 |
| Pollbooks | 105 | 888 | 0.081 | 52.6 |
| Words | 112 | 425 | 0.068 | 41.6 |
| Power | 4941 | 13188 | 0.001 | 390.4 |

\*Each label represents a graph



As expected, we found that when the average number of iterations to destruction is normalized through dividing by the amount of nodes in the graph, this measure and the graph density share a strong upward trending correlation. That is, the greater the density of the graph, the greater the number of iterations it takes to destruct the graph, generally. And while this information may not sound interesting or unobvious when communicating to students of graph theory, this may not always be the case when speaking to those planning for real-world situations.

It may seem initially intuitive, for example, in the case that one wishes to shield an important facility from potential threat, to keep any connections going in and out of that facility to a minimum. Indeed however if this is the case, it is likely that such a facility would need to have at least one connection to a hub of higher importance (a node with high betweenness centrality), and with one connection to one important node, in our algorithm the facility is likely to be disconnected quickly.

Instead, our data suggests that the best way to protect an important node is to actually hide this node in a sea of seemingly important nodes, as measured by betweenness centrality. That is, if every node is near equally as important as the next in terms of betweenness centrality, then it becomes very difficult for an adversary to pinpoint a capital node in terms of this measure.

As long as the adversary does not have the ability to distinguish between ‘real’ important connections and ‘fake’ important connections, adding in as many fake connections as possible should always be preferable. A good example to imagine this would be in the planning of a power grid network. If the building nation was concerned about a possible attack to take this network down, and the potential adversary has access to a graph of said network, it would be in the building nation’s best interest to make each of the nodes in the graph (power poles) as equal as possible in terms of number of connections. In the absence of this ‘faking’ process, the more important nodes would be easily identifiable by a greater betweenness centrality, and therefore would also be much more likely to be targeted by an attack. By adding fake connections, the building nation engages in the process of ‘masking’ the important connections by artificially blending the important nodes in with the non-important nodes.

\*The software for this project can be found at [3].

Bibliography

[1]

A. Kurdia, “CentralityComputer,” *Centrality Computer*, 23-Aug-2008. [Online]. Available: https://trac.project.fit.cvut.cz/w20ss11t103t8/browser/src/java/gexf/tools/centralitycomputer.java?rev=66. [Accessed: 10-Oct-2016].

[2]

D. J. Watts and S. H. Strogatz, “Collective dynamics of /’small-world/’ networks:Abstract: Nature, “*Nature*, vol. 393, no. 6684, pp. 440-442, Jun. 1998. [Online].Available:http://www.nature.com/nature/journal/v393/n6684/full/393440a0.html.Accessed: Oct. 1, 2016.

[3]

K. Hagood, "Hagoodkk/SNAProj," in *GitHub*, 2016. [Online]. Available: https://github.com/Hagoodkk/SNAProj.git. Accessed: Nov. 02, 2016.

[4]

M. Newman, "Network data," in *http://www-personal.umich.edu/~mejn/netdata/*, 2013. [Online]. Available: http://www-personal.umich.edu/~mejn/netdata/. Accessed: Dec. 3, 2016.

[5]

gephi, "Gephi/gephi," GitHub, 2016. [Online]. Available: https://github.com/gephi/gephi/wiki/Datasets. Accessed: Dec. 3, 2016.

[6]

S. Rizvi, "Gephi/gephi," in *https://github.com/rizvish/SNAproject*, GitHub, 2016. [Online]. Available: https://github.com/gephi/gephi/wiki/Datasets. Accessed: Dec. 3, 2016