**EE144: Pandemaniac Report**

**“Engineers at Network”**

*Akshta Athawale, Mannat Singh and Miguel Aroca-Ouellette*

**1. Overview**

For the Pandemaniac competition, our team took a multi-faceted approach wherein we designed different seed generation algorithms for the different portions of the mini-project. The first seed generation algorithm was aimed at scoring well in general, the second and third on beating the TAs and the fourth for scoring well against multiple TA players.

We have three members in our team, Akshta Athawale, Mannat Singh and Miguel Aroca-Ouellette. Akshta was in charge of developing the “Betweenness” algorithm and improving the “Max Degree” algorithm. Mannat handled implementation of the “Clustering” algorithm. Miguel wrote the base class, network simulation wrapper, network visualization, game tree and the “Degree Killer” algorithm. However, there was significant collaboration on most tasks from every member of the team, particularly in the discussion, design and analysis of algorithms and their results.

**2. Max Degree**

The Max Degree algorithm was our first approach at generating seeds for the provided networks. It does so by iterating through the adjacency lists and finding the top nodes with the highest degree, where is the number of seeds for the specified graph.

The main issue with this approach is that it will never beat *TA\_degree*, and will most likely do quite poorly against other teams, who – particularly at the early stages of the mini-project – will be using a similar technique to generate seed nodes. If more than one team uses this technique to generate seed nodes, then their nodes will simply cancel out and any teams using the same technique will have no seed nodes. In order to solve this problem, we chose to introduce some randomization of the top degree nodes. This was done finding the top nodes with the highest degree, where and is defined as above. Then, for each game round a random subset of size was chosen from the complete set of seed nodes. This allowed us to avoid complete collisions, while still selecting favourable nodes. Using this approach our team managed to place 2nd on day 3 and beat *TA\_fewer*.

**3. Betweenness**

When we could not beat the TA graphs using the Max degree approach, we thought using Betweenness centrality along with degree centrality might improve our results.

For this approach we chose more number of nodes than required, i.e. If number of seeds are required, then we chose nodes with top values of degree centralities and then chose the top nodes among these based on their betweenness centralities. We expected this approach to work better because clearly among the top degree nodes the ones with higher betweenness centralities should cover a larger portion of the graph and faster. For example, in the case when a cluster is connecting two or more clusters, we would want to cover the connecting cluster instead of one of the other clusters and this approach would make that possible.

The issue with this technique was that for the graphs provided, most of the max degree nodes were the also the ones with maximum betweenness centralities and hence this did not change our results by much.

**4. Degree Killer and Game Tree**

Unfortunately, by the 3rd day of the competition we had yet to be successful in beating *TA\_degree* and *TA\_more*. Since *TA\_more* was simply a bonus, we first focused on developing an algorithm to beat *TA\_degree*, which we named Degree Killer.

Since *TA\_degree*’s strategy was known and we had access to the simulation library, we realized that we could use brute force to generate a set of seed nodes to beat *TA\_degree*. However, we obviously could not iterate through all possible seed combinations within the set of available nodes, so we first had to prune the number of candidate nodes to an acceptable size. Our intuition was that most of the desirable nodes would have a high degree, and would be connected to other nodes of high degree; a loose measure of centrality. For the Degree Killer algorithm, we chose to look at the intersection between the top high degree nodes and the neighbours of the top high degree nodes, where was an empirically determined number balancing computation time and combination variety. Using this set, we then generated every possible seed node combinations and competed them against our Max Degree algorithm with no randomization (equivalent to *TA\_degree*’s strategy). Once we found a set of seed nodes which beat the Max Degree algorithm, we then use that set of nodes to compete against *TA\_degree*, guaranteeing a victory. For a graph with 10 seed nodes our algorithm had to check at most 286 possible node combinations, which was easily done within the 3-minute time limit for the competition. This algorithm allowed us to beat *TA\_degree* on day 4.

**5. Clustering**

We realised that using just degree and betweenness centralities led us to pick seed nodes in clusters which were really dense. If we were just competing with one team with the same (or less) number of seeds as us, these approaches worked well. But when there were multiple players (or when we were playing against *TA\_more*), due to the fact that high degree seed nodes were picked by some team with a very high probability, we would some times end up with very few seed nodes. To address this, we tried to utilise clustering.

The first algorithm we tried was K-means, but it takes a feature matrix as input, not an affinity (adjacency) matrix – this led to high runtimes if we used the adjacency matrix simply as a feature matrix. Since spectral clustering takes adjacency matrices as inputs and the graphs were not too complex, we settled on using this approach for cluster detection. Choosing the number of clusters was difficult, but based on a qualitative inspection of graph visualizations and competition results we settled on a using 5 clusters.

Clustering gave us tags which told which cluster each node was part of. After looking at the clusters, we realised that quite a few times the high degree nodes were all just part of one cluster, which would very often not be the largest cluster. In addition, there would be often one cluster which would be quite large, but which had a collection of nodes with very small degrees – this represented a poorly connected cluster and such nodes were most likely grouped together simply since they didn’t belong to any other clusters. As such it was undesirable to place seed nodes within this cluster.

Once the clustering labels had been identified, we tried various approaches to generate the seed nodes. The first approach we tried was to select high degree nodes from each cluster – this worked very poorly, as we weren’t able to “take over” even one cluster in reasonable time, and often all our nodes would be eliminated in a few iterations.

We then focused on ensuring that we maximized our epidemic spread over a few clusters. We used the average degree in a cluster to determine its density, and the number of nodes inside it to determine its size. After trying various combinations, we ended up with the following criterion for cluster selection:

* Cluster the graph into 5 clusters.
* Find the two densest clusters, based on the average degree.
* Choose the cluster with the highest number of nodes, out of the two densest clusters.

After this, we again randomised selection of the highest degree nodes, but only from that cluster. Another improvement we applied was to use DegreeKiller, but with our candidate nodes restricted to being within the specified cluster.

With this, we were able to repeatedly beat *TA\_more* for all the days. We also tested this approach in round 5, where out of the 4 graphs with 8 players, we got 20 points for two graphs, 6 points for one, and 1 point for the last.

**6. Visualization and Testing**

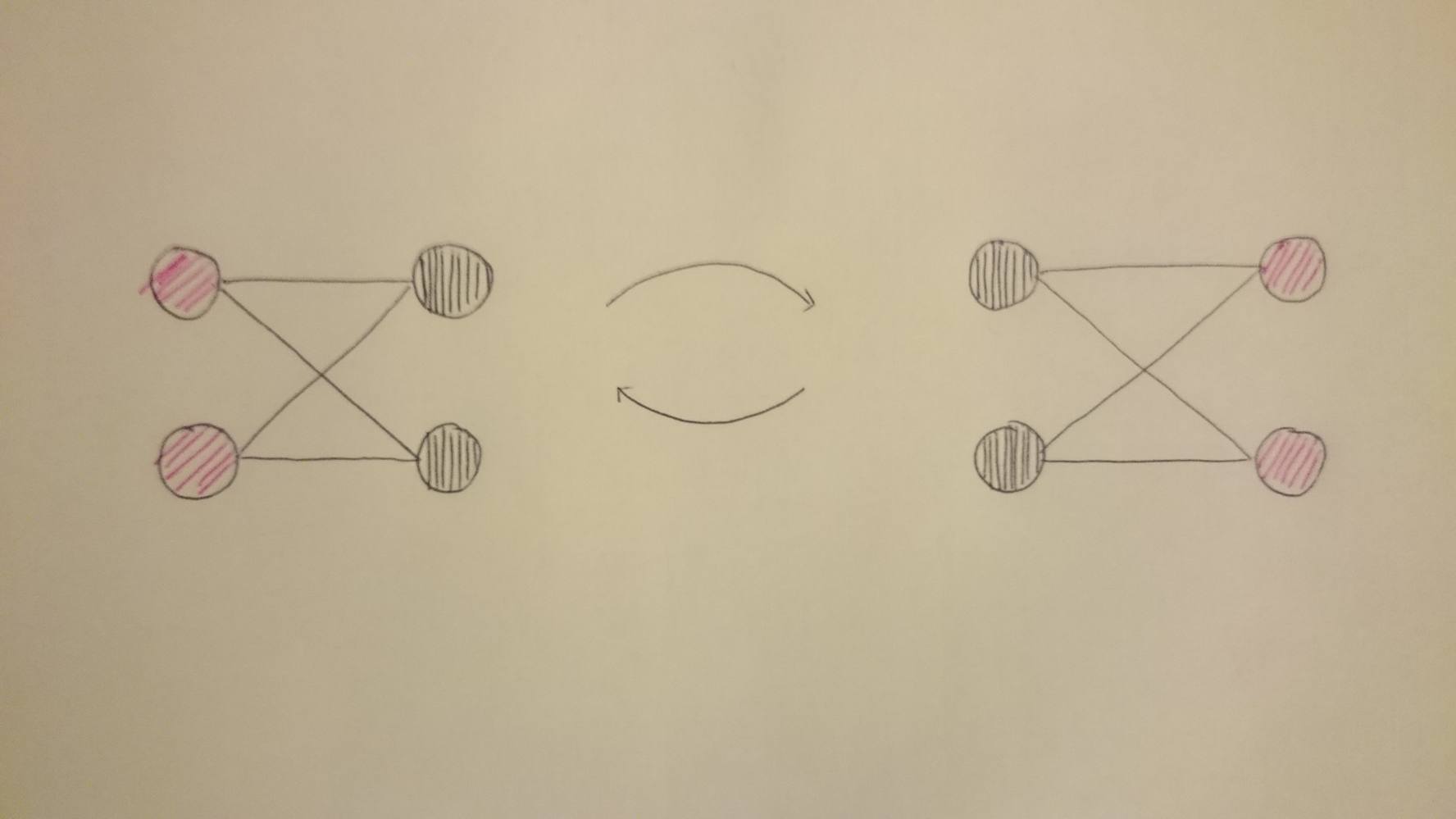
In order to gain a better understanding of how our algorithms were performing against each other, the TA’s and the other teams we developed our own pandemic visualizations. Using *networkx*, *matplotlib* and the provided simulation package we wrote methods which, given a list of competing seed nodes, allowed us view both the initial placement of seed nodes and the final result of a competition. We also wrote abstractions which allowed us to view the results of the online competition after having downloaded the appropriate seed and graph files. These visualizations not only allowed us to gain a better intuitive understanding of how our, and other, algorithms were operating and the limitations of different approaches, but also made it much easier to debug our logic.

The visualization’s also made it clear that the dominant algorithm depended very much on the graph structure, and since we had several algorithms suited to a variety of graphs we reasoned that we would need to select the best algorithm given a graph. To do this we wrote a game tree function which would compete sets of seed nodes against each other in round-robin style and would return the most successful set of seed node based on the assignments scoring metric. This allowed us to combine the different approaches of our algorithms to generate a single, best set of seed nodes.

**7. Conclusion**

Overall, our team was successful in beating all 3 TA milestones and placed strongly in the 3 practice rounds in which we participated. We achieved this by using a variety of algorithms, each designed either to beat a TA milestone, or produce favorable results in the class rankings. By developing a visualization and testing system we were able to analyze and improve our algorithms, as well as select the best algorithm for a given graph.

**8. Counter Example**



Clearly the above graph and initial coloring yields epidemic colors which do not converge.

**9. Suggestions for Next Year**

Graph visualization would, of course, be helpful, but it would perhaps be even more helpful if it allowed visualization for every iteration of every round. This way you could see the spread of each team’s node at every iteration and gain a better understanding of how the seed generation algorithms interact with each other.

Our team also felt that the optimal seed nodes depended so strongly on the graph that it was difficult to find a strategy which worked well on all graphs. The game tree helped a little in this regards, but perhaps for the competition rounds it would make sense to only work with one type of graph (of varying size and somewhat randomized of course).