CS/EE 144 Pandemaniac Report

Team kraftwerk
Fabian Boemer, Jessica Li

February 28, 2016

1 Introduction and Overview

1.1 Introduction

The goal of pandemaniac is to compete against other teams to infect as many nodes of a given graph as possible. Each graph is undirected and unknown before the game begins. The only influence a team may have on any game is the selection of seed nodes. Teams develop methods to select the most "influential" seed nodes without overlapping, since nodes selected by more than one team is nullified.

After seed nodes for both teams are determined, each graph is iterated upon until the coloring converges, or for 100-200 times otherwise. The graph provided for this project all converged, but some, as described in Section 1.2 do not. In each iteration, nodes are colored, or recolored, based on the coloring of itself and its direct neighbors. A colored node contributes 1.5 votes to that color. A neighboring node contributes 1 vote to its color. An uncolored node does not get a vote. If a color wins a strict majority of the votes, then the node is converted. Otherwise, no change is made.

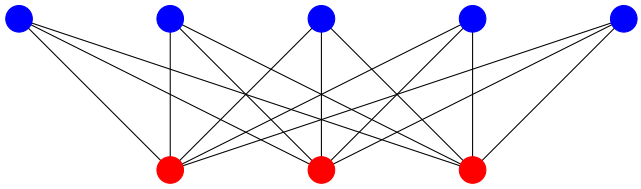
Our team's efforts were directed towards beating the TA's seed selection algorithms during "regular season." In this report, TA-less refers to the selection of fewer seed nodes than allotted; TA-degree refers to the selection of seed nodes based on highest degree; TA-more refers to the selection of more seed nodes than allotted. The seed selection criteria for TA-less and TA-more are not known.

All code was written in Python and utilized various Python packages, such as Networkx [2] , Community [3], Matplotlib [4], and JSON.

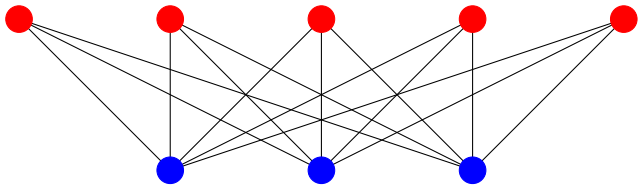
1.2 Counterexample

In some cases, a set of initial colorings may result in a game which fails to converge. Consider the complete bipartite graph, $K_{m,n}$, $m, n \geq 2$ with disjoint nodesets U, V and initial colorings red $A = U$, blue $B = V$. Under this inial coloring, the direct neighbors of each node are nodes of the other color. In other words, nodes of the same color are not direclty connected.

In the first iteration, A-colored nodes will be converted to the color of the nodes in V and B-colored nodes will be converted to the color of nodes in U. For all following timesteps, the colorings will continue to alternate between the bipartite sets. Figure 1 shows the behavior of this game, which fails to converge.



(a) Graph at $t \mod 2 = 0$



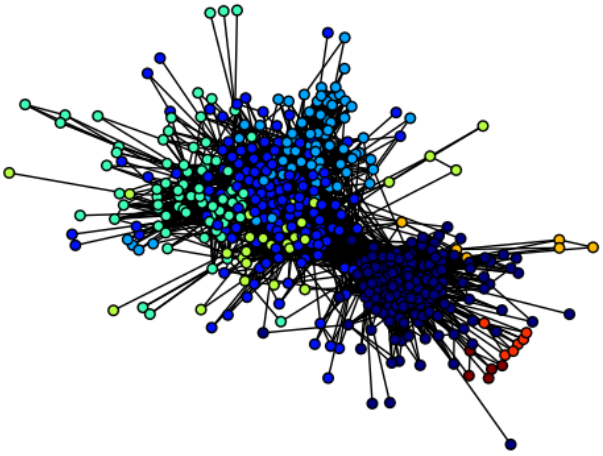
(b) Graph at $t \mod 2 = 1$

Figure 1: Example of a graph and initial coloring that fails to converge

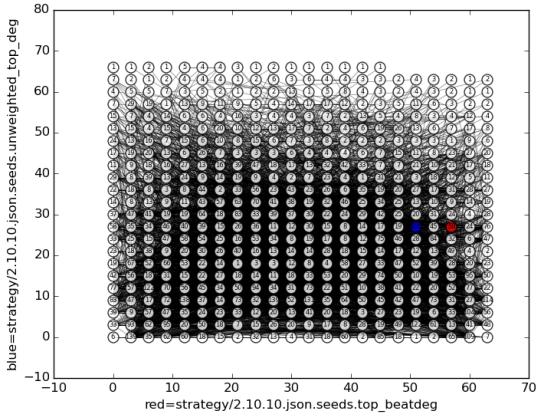
2 Visualization

Visualization was an important component of this project, useful for better understanding the graph's structure and how color propagated from selected seed nodes. We explored several built-in and custom visualizations provided in Networkx.

We attempted to visualize the clustering/community structure of the graph, using Networkx's visualization and Community's built-in partition functions. Nodes were sorted and colored based on community. The graph was then displayed using Networkx's "spring layout", which uses Fruchterman-Reingold force-directed algorithm. Nodes modeled as having repulsive forces and edges are modeled as spring. A balance of these forces allows the graph to be laid out with as few overlapping edges as possible. Figure 2 shows a coloring of graph clusters. Clusters are distinguishable, though not clearly separable.



(a) Spring layout of graph clusters.



(b) Grid layout

Figure 2: Visualization of graphs 2.10.32 and 2.10.31

To see color propagation, we generated our own visualizer, in place of the non-functioning one. One criterion for visualizing color progression was maintaining a consistent graph between iterations. A circular layout was not instructive because nodes were too close together. We implemented a custom grid layout, as in Figure 2 where each node is labeled with its degree. The grid layout failed to visually capture the structure of the graph, but was useful for viewing the progress of color cascades. We may reasonable indentify clusters based on on the color propagations.

We only used visualization for offline learning. A more complete approach would be to human input to select seed nodes by "eyeballing" important nodes in clusters. However, we did not consider our visualization displayed graph structure well enough to implement this approach.

3 Algorithms

3.1 Day 1: Cancellation

The first approach was a naive cancellation approach designed to beat TA-degree. Knowing exactly how seed nodes were selected, we sought to negate all but the lowest-degree seed node and conquer the highest degree node. Algorithm 1 gives the pseudocode for the cancellation approach.

Algorithm 1 Cancellation Algorithm

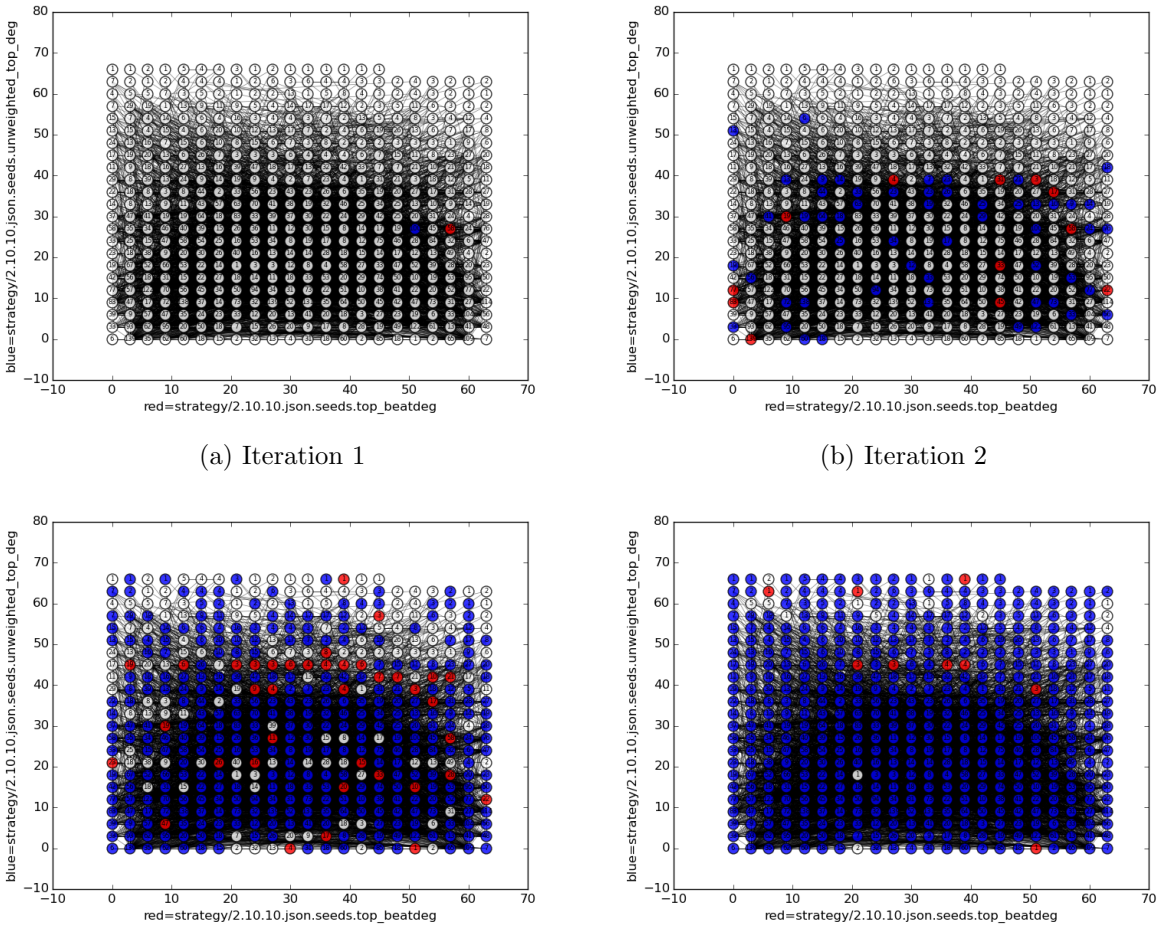
```
procedure CANCELLATION(G, n)
    Order the nodes by degree centrality
    Select the n − 1 nodes with highest degree centrality
    Select the previously-unselected node adjacent to the vertex of highest-degree.
    return selected nodes.
```

Time constraints prevented us from testing this strategy before the Day 1 run. We retrospectively visualized the results from the Day 1 run against TA-degree, shown in Figure 3. In Iteration 1, 3a, we notice only one node of each color is selected, by construction. This implies all other nodes were negated. In Iteration 2, 3b, we claimed the highest degree node in this graph, the red node adjacent to the bottom-left corner with degree 139. However, the TA's blue seed node had a higher degree than our seed node and claimed several blue nodes. In Iteration 3, 3c, we see the highest-degree node failed to convert many nodes. The TA already has spread to enough nodes that converting any new nodes is difficult. By Iteration 4, 3d, it is clear the blue nodes have won. Thus, our cancellation approach moves too slowly. Controlling the highest-degree node at Iteration 2 is not enough to win against TA-degree.

As simulated, we did not beat TA-degree on Day 1. We did, however, beat TA-fewer. Though not our intention, this was not unexpected, since the set of $n - 1$ nodes with high degree centrality is a decent selection that conquers rapidly.

3.2 Day 2: Centrality Measures

On Day 2, we focused on developing a different strategy to beat TA-degree. We explored the several centrality measures built into NetworkX, including betweeness, current flow betweenness (cfbet), closeness, and eigenvector centrality. First we tried taking top n most-important nodes for each centrality measure ranking. We excluded degree-centrality since TA-degree uses degree centrality.



(a) Iteration 1

(b) Iteration 2

(c) Iteration 3

(d) Iteration 4

Figure 3: Visualization of the Day 1 run against TA-degree. Our team's nodes are red; the TA's nodes are blue. The numbers in the nodes indicate the node degree.

We modified "sim.py" to simulate each selction strategy against TA-degree. Table 1 shows the results of our simulations

This new method was tested on Day 1's TA-degree graph. We see in Table 1 that closeness and eigenvector centrality consistently outperformed TA-degree. For our submission, we set up a local pipeline to generate seeds for current flow betweeness, closeness, and eigenvector centralities, and simulate TA-degree against each centrality measure. We expeceted at least one of the three strategies to beat TA-degree on graph 2.10.11, based on the performances on graphs 2.10.10, 2.10.20, 2.10.30.

Unfortunately, during submission, we simulated narrow losses against TA-degree with closeness and eigevector centralities, with a heavy loss with current-flow betweenness. The simulation was correct, as we failed to beat TA-degree.

We spent some time trying to visualize the structure of the graphs. Unforunately, the visual-

Day	Graph	Centrality measure	Strategy count	TA-degree count	Ratio
1	2.10.10	betweenness	18	482	0.04
	2.10.10	cfbet	491	9	0.99
	2.10.10	closeness	386	113	0.77
	2.10.10	eigenvector	394	106	0.79
	2.10.20	betweenness	43	455	0.08
	2.10.20	cfbet	43	455	0.08
	2.10.20	closeness	452	45	0.90
	2.10.20	eigenvector	429	69	0.86
	2.10.30	closeness	475	22	0.96
	2.10.30	eigenvector	469	28	0.94
2	2.10.11	cfbet	31	468	0.06
	2.10.11	closeness	231	263	0.47
	2.10.11	eigenvector	239	256	0.48
	2.10.21	closeness	277	216	0.56
	2.10.21	eigenvector	206	283	0.42
	2.10.31	closeness	35	456	0.07
	2.10.31	eigenvector	28	464	0.056

Table 1: Summary of local testing results for each centrality measure for Day 2.

izations produced by NetworkX's "spring layout" were inconsistent between successive drawings, and the remaining visualization formats did not expose the structure of the graph.

We tried clustering the graph using NetworkX's, "k_components = apxa.k_components(G)." For 500-node graphs, the clustering took 129 seconds, which was deemed too long for the 3 minute deadline.

3.3 Day 3: Monte-Carlo

Having failed to beat TA-degree, we increased the scope of our approach. We explored several more unfamiliar measures of centrality, and began used multiple centrality measures for selecting a set of seeds. We included Katz centrality and dispersion centrality. Katz centrality solves a system of linear equations for the centrality of a node relative to that of its neighbors [2]. Dispersion centrality measures how close are mutual friends between pairs of nodes in the graph [2] In some sense, it is another measure of "betweenness."

Because each of these centrality measures took only a few seconds, we were able to use our time more completely through Monte-Carlo simulations. Algorithm 2 is the pseudocode for this approach.

This was a non-disciminative approach, which does not distinguish between the efficacy of each centrality, nor the complementary advantages of each centrality measure. For example, nodes with high betweenness measure are likely prevent cascades from spreading, while nodes with high closeness centrality spread quickly to all nodes in a cluster. Nevertheless, this approach successfully

generated several strategies simulated to win against TA-degree. Having not yet beat TA-more, we varied our final output among the top-10 best-performing strategies simulated to beat TA-more. This provides more variance in the seeding, and makes the seeding more robust against an adverserial TA seeding.

As simulated, we successfully beat TA-degeree with this approach. However, we still failed to beat TA-more.

Algorithm 2 Promising Algorithm

procedure PROMISING(G, n)
    Order the nodes by degree centrality, eigenvector centrality, Katz centrality, closeness centrality, dispersion centrality.
    Initialize dictionary d with keys all the nodes and values as 0
    for each centrality measure **do**
        for node in top n ranking **do**
            Let d[node]$+ = n-$ ranking.
    Sort d descending by value and choose 12 highest-value nodes as 'promising'
    for each of $\binom{12}{10}$ combinations of 10 seed nodes **do**
        Simulate the seed against TA-degree
    Order seeds by number of nodes claimed against TA-degree
    while Final Seeding not complete **do**
        for the top ten seeds ranked by performance **do**
            populate Final Seeding with the ten seeds
return Final Seeding.

3.4 Day 4: Blending

The remainder of regular season was focused on beating TA-more. We refined our idea of blending promising nodes. We took the n most promising nodes based on degree centrality, betweenness and eigenvector centrality. We reasoned that nodes with high degree centrality would effectively conquer clusters, while nodes with high betweeness would effectively blockade cascasdes from proceeding between clusters. Up to this point, eigenvector centrality performed quite well and was computed very rapidly, so we decided to include the nodes with high eigenvector centrality as well.

Algorithm 3 Hybrid Algorithm

procedure HYBRID(G, n)
    Order the nodes by degree centrality, eigenvector centrality, Katz centrality, closeness centrality, dispersion centrality.
    Generate the set s of top-n ranked nodes in each centrality measure
    Generate 100 random selections of n nodes from s.
    for each of 100 seed node selections **do**
        Simulate seed nodes against TA-degree with $1.2n$ nodes.
    return best-performing set of seed nodes.

On the Day 4 run, we still failed to beat TA-more. Thus, the Monte-Carlo generation of "important" nodes failed to quickly cover enough variation in seed nodes to beat TA-more

3.5 Day 5: Clustering

On Day 5, we tried using the Louvain method [1] based function provided in the Community package to partition the graph. This approach was much faster than than the previous approach using "k_components = apxa.k_components(G)." The size of the clusters varied from less than 10 to over 100 for a 500-node graph, and were not disjoint. We adapted previous approaches to clustering, described in Algorithm 4.

Algorithm 4 Cluster Algorithm

```
1: procedure HYBRID(G, n)
2:     Cluster G using Louvain method
3:     for each cluster c do
4:         Use promising(c, n) to generate 'most-important' node within the cluster.
5:     for each of 3 populating strategies do
6:         Initialize empty FinalSeeding list
7:         while FinalSeeding length less than n do
8:             for each cluster c do
9:                 Insert most-important node(s) from c not yet in FinalSeeding
10:            Simulate FinalSeeding against TA-more Day 4
11:    return best-performing FinalSeeding
```

For each cluster, we explored three methods of choosing the number of most-important node(s) selected. For a given cluster c, we populated FinalSeeding with the number of nodes proportional to one of:

(1) 1 (each cluster has the same number of nodes)

(2) the total degree of nodes in c

(3) the number of edges in c

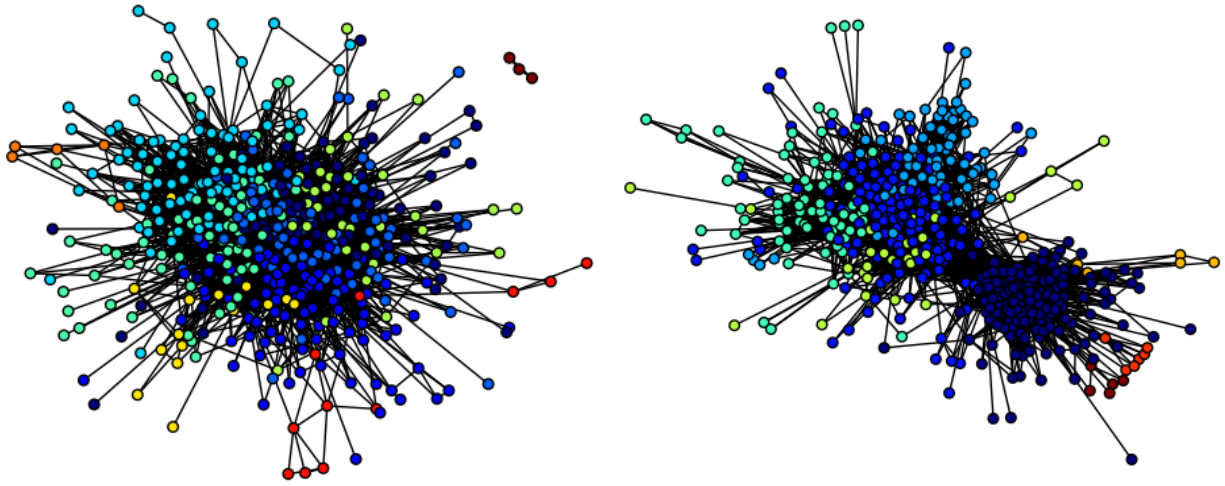
We simulated the results against the nodes picked by TA-more on previous days. Our findings are summarized in Table 2.

Graph	Strategy	Proportion Nodes Claimed	Games won
2.10.30	(1)	0.13	5
2.10.30	(2)	0.25	12
2.10.30	(3)	0.07	1
2.10.31	(1)	0.04	0
2.10.31	(2)	0.06	1
2.10.31	(3)	0.04	0
2.10.32	(1)	0.60	47
2.10.32	(2)	0.21	4
2.10.32	(3)	0.21	7

Table 2: Summary of local testing results using clustering approach for Day 5.

We were able to win on graph 2.10.32, but did very poorly on graphs 2.10.30 and 2.10.31. We visualized and investigated the structure of the graphs to determined why this discrepency occurred.

The visualization in Figure 4 between graphs 2.10.31 and 2.10.32 suggested 2.10.32 was more clearly clustered than 2.10.31, suggesting our approach (1) works well for clustered graphs. This makes sense for clusters of roughly equal size. Nevertheless, we see 2.10.31 has a separate cluster of only 3 nodes, which is likely not worth placing a seed on. In retrospect, perhaps our strategies were placing too much weight on small clusters. Rather, perhaps, we should have chosen only nodes from the largest cluster.



(a) 2.10.31

(b) 2.10.32

Figure 4: Visualization of graphs 2.10.31 and 2.10.32

In Table 3 in Section 5, we see graph 2.10.32 has higher average and overall clustering coefficient, compared to graph 2.10.30, supporting the hypothesis that 2.10.32 is better clustered. 2.10.32 also a higher Louvain modularity, which means the Louvain method of clustering would produce better partitions on this graph. These results might suggest our approach (1) might only work well for graphs with these properties.

Under time constraint, we decided to use the number of nodes proportional to the number in each cluster, since this was the only winning approach we discovered. Unfortunately, this failed to beat TA-more.

4 Conclusion

We explored various methods of selecting seed nodes and successfully beat TA-fewer and TA-degree. We did not manage to beat TA-more during regular season. Due to other commitments and time constraints, we did not actively participate in the tournament.

We considered a few other approaches which were not implemented due to time-constraints. One approach was to explore parallel computation of centrality measures, which would have proven useful on the larger graphs. A different approach was using python's Gambit library [5] to find Nash equilibria for a 2 (or more) player game. Given a graph $G = (V, E)$, we would construct an $n \times n$ zeros-sum payoff matrix, where $n = |V|$ with the payoff as some measure of importance, such as some combination of centrality measures. The multi-player Nash equilibrium would have generated

better strategies for the tournament.

We learned about other measures of centrality and exploring different combinations of centrality measures to "cover up" weaknesses. We learned to implement various Python packages associated with analyzing graphs and the runtime differences between functions. Our rudimentary implementation did not show clusters well, but did allow us to get a sense of spread of each color and graph structure. We learned the importance of documenting methods and results for sound scientific method as well as the importance of trying a broad set of approaches.

5 Appendix

Graph	Average Clustering Coefficient	Overall Clustering Coefficient	Connected Components	Maximal Diameter	Average Diameter	Partitions	Louvain Modularity
2.10.30	0.44	0.28	2	6	2.61	7.0	0.35
2.10.31	0.46	0.28	2	7	2.65	10.0	0.34
2.10.32	0.51	0.36	1	6	2.83	8.0	0.5
2.10.33	0.47	0.3	1	7	2.61	12.0	0.34
2.10.34	0.48	0.27	1	7	2.83	8.0	0.47
2.10.20	0.42	0.29	1	7	2.78	9.0	0.4
2.10.21	0.53	0.36	2	7	2.87	11.0	0.48
2.10.22	0.47	0.29	1	7	2.58	6.0	0.38
2.10.23	0.47	0.3	1	6	2.53	6.0	0.36
2.10.24	0.48	0.31	1	6	2.64	9.0	0.36
2.10.10	0.48	0.31	1	6	2.55	7.0	0.4
2.10.11	0.51	0.32	1	8	2.68	8.0	0.33
2.10.12	0.52	0.36	1	7	2.93	5.0	0.45
2.10.13	0.5	0.33	1	6	2.95	9.0	0.49
2.10.14	0.5	0.32	1	6	2.75	9.0	0.46

Table 3: Submission graph information

References

[1] Fast unfolding of communities in large networks, Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, Renaud Lefebvre, Journal of Statistical Mechanics: Theory and Experiment 2008(10), P10008 (12pp)

[2] https://networkx.github.io/

[3] https://pypi.python.org/pypi/python-louvain/0.3

[4] http://matplotlib.org/

[5] http://www.gambit-project.org/gambit14/pyapi.html