CMS/CS/EE144 **Team Kaigoo**

Miniproject : Pandemaniac

02/25/2018 Hangwen Lu, Guanya Shi, Botao Hu

1 Team Member & Work Split

- **Team Name**

  Team Kaigoo

- **Group Members**

  Hangwen Lu
  Guanya Shi
  Botao Hu

- **Division of Labor**

  Hangwen Lu: Collaborate on generating basic strategies and analyze results of previous games. Research on graph partition algorithm and Laplacian embedding theory.
  Guanya Shi: Collaborate on generating basic strategies. Did thorough local test on our proposed strategies. Came up with game theory thoughts and implementations.
  Botao Hu: Collaborate on generating basic strategies. Built up and maintained the code structure, I/O and helped on realizations of centrality measures and clustering methods.

2 Counterexample

Based on the rule of the spread of epidemics, we can construct the following counterexample which will not converge.

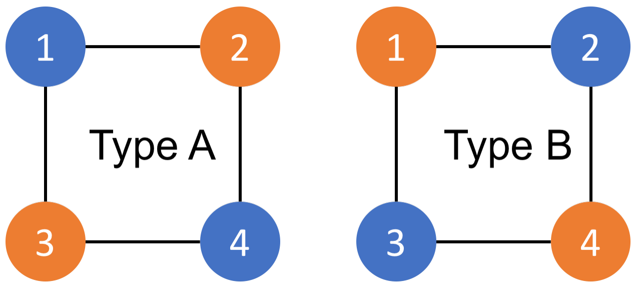

Figure 1: Non-convergence graph example

For each node in this graph, two neighbors of it are both enemies, so it will switch to the enemy's color. For example, node 1 in type A has 1.5 votes for color blue and 2 votes for color orange, thus it will switch to orange. Because updates of all nodes are simultaneous, this graph will oscillate between type A and type B.

3 Clustering Methods

- **Graph Laplacians theory**

  We incorporated spectral clustering methods using the adjacency matrix in the largest connected subgraph of the network G.

  The spectral graph theory studies the properties of graphs via the eigenvalues and eigenvectors of their associated graph matrices: the adjacency matrix and the graph Laplacian and its variants.

  For a graph with n vertices, the entries of the $n \times n$ adjacency matrix are defined by:

$$\mathbf{A} = \begin{cases} a_{i,j} = 1 & \text{if edge } e_{i,j} \text{ exists} \\ a_{i,j} = 0 & \text{if edge } e_{i,j} \text{ doesn't exist} \end{cases}$$

  $\mathbf{A}$ is a real-symmetric matrix: it has n real eigenvalues and its n real eigenvectors forming an orthonormal basis. The Laplacian matrix is defined as:

$$\mathbf{L} = \mathbf{D} - \mathbf{A}$$

  where $\mathbf{D}$ is a diagonal matrix with $d_{i,i}$ = degree of node i. $\mathbf{L}$ is symmetric and positive semi-definite and has n non-negative, real-valued eigenvalues $\lambda_1 \leq \lambda 2, \leq ... \leq \lambda_n$. Their eigenvectors are $v_1, v_2, ..., v_n$.

  For a connected graph, $\lambda_1 = 0$ with $v_1 = \mathbf{1}_n$. The first non-null eigenvalue λ_2 is called the Fiedler value. The corresponding eigenvector v_2 is called the Fiedler vector. The Fidler vector is a good tool for spectral bi_partitioning and analysis the clustering properties.

- **Laplacian Embedding**

  To visualize the clustering properties of the graph, we can embed the graph in a 2-dimensional Euclidean space by taking the first 2 eigenvectors of non-null eigenvalues and generate a 2-D spectral distribution all nodes. We use `sklearn.manifold.SpectralEmbedding` package to generate the spectral distribution.

  An example embedding is shown below (graph 8.20.5):

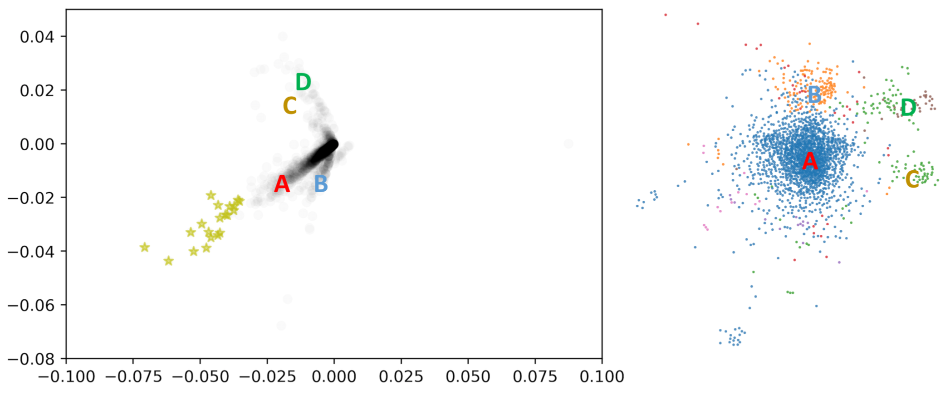

Figure 2: Laplacian spectral distribution of Graph 8.20.5 and network cluster distribution. The yellow stars stands for nodes with top degree centrality

- **Decode network clustering properties with Laplacian spectrum**

  In Laplacian spectrum, the $x - y$ axises represent the 1st and 2nd order spectral information (distributed around 0 and spread in range[-1,1]). As shown in figure 2, we are able to "guess" the clustering properties of the network based on this spectral distribution. Following are some criteria we conclude based on our experience from all the graphs provided.

  - Each direction(branch) represents a cluster.
  - The left-bottom most branch(also the most dense one) stands for the dominant cluster. and other clusters spread in different directions.
  - Nodes lying away from (0,0) are those in the center of clusters and has higher importance in the spreading model, while nodes around center(0,0) are surrounding vertices with less importance.

  Generally, there are two kinds of networks in the game, besides those with dominant cluster and small surrounding clusters like 8.20.5, some networks has 2 clusters (for example, 8.10.5) with similar size, shown in figure 3.

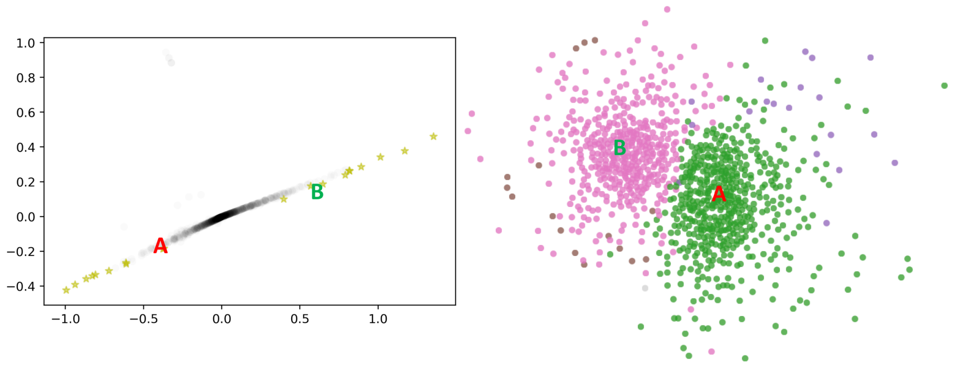

Figure 3: Laplacian spectral distribution of Graph 8.10.5 and network cluster distribution.

- **Partition based on spectral clustering**

  Partition is important for two reasons:

  - Some graphs are huge, partition can reduce the complexity
  - We can focus our seeds on single cluster and get higher chance to occupy it.

  We tried using spectral cluster method to partition the graph. Directly clustering from adjacency matrix doesn't give us satisfying result, since the matrix is huge and complex. Instead, we partition on the Laplacian spectrum since the clusters are separated spatially. We choose 'nearest_neighborhood' method and partition them into two sub graphs using `sklearn.cluster.SpectralClustering`. Result shown in figure 4

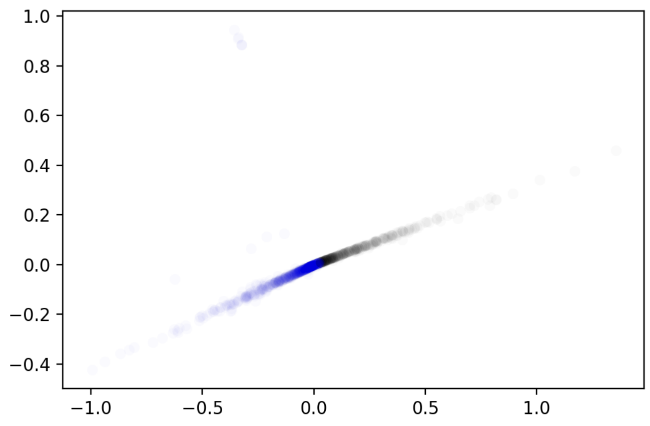

Figure 4: spectral partition of graph 8.10.5

- **Other partition method (Louvain algorithm)**

  We also tried to compute the partition of the graph nodes using the Louvain heuristices, which is the partition of highest modularity [3]. We directly used `best_partition` function in package `community`. Unfortunately, the partition results didn't perform very well so finally we used spectral clustering methods instead.

4 Centrality Measures

How to measure the "importance" of nodes is the key to this problem. To discover how to measure it appropriately, we studied on the following four centrality measures.

- **Degree centrality** The degree centrality for a node v, $D(v)$, is the fraction of nodes it is connected to, which is proportional to the degree of v. Nodes with higher degree will be more important. We used the function `degree_centrality` in python package `networkx`. The complexity of this centrality is basically $O(E)$, where E is the number of edges in a graph.

  Note that computing degree centrality is pretty cheap because the degree information is already stored in a `networkx` graph.

- **Closeness centrality** Closeness centrality of a node v, $C(v)$, is the reciprocal of the sum of the shortest path distances from v to all $n-1$ other nodes. Since the sum of distances depends on the number of nodes in the graph, closeness is normalized by the sum of minimum possible distances $n-1$. $C(v)$ can be expressed as:

$$C(v) = \frac{n-1}{\sum_{u=1}^{n-1} d(v,u)},$$

  where $d(v,u)$ is the distance between v and u. We used the function `closeness_centrality` in python package `networkx`.

  Note that computing closeness centrality is much more expensive than computing degree centrality. Because **Dijkstra's algorithm** is applied to compute $d(v,u)$, with the running time $O(E \log n)$, the total running time to compute a graph's closeness centrality would be $O(n^2 E \log n)$ [1].

- **Betweenness centrality** We used the `current_flow_betweenness_centrality` funtion in the `networkx` library. In contrast to shortest path betweenness, current flow betweenness centrality uses an electrical current model to measure the node's importance in information spreading process[4], which we thought would be a good measure. The time complexity of this method is, according to the proposer, $O(T + nE \log n)$, where T is the time to compute inverse Laplacian of adjacency matrix. If the matrix is sparse we can reach $T = O(nE\sqrt{k})$, where k is the condition number of Laplacian matrix.

  Note that comparing with others, this measure is a particularly time consuming one. If we are given a graph with more than 10,000 nodes, this measure takes at least 10 minutes to finish. Fortunately, we later on found some clustering methods which helps us to get smaller graphs and make the algorithm faster, but this measure did not perform very well so we treated it as a reference.

- **Mixed centrality** From the definition of degree centrality and closeness centrality, we can find that for each node, the degree centrality measures how many other nodes it can influence in one iteration, and on the other hand, the closeness centrality measures how quickly it can influence to other areas.

  Therefore, here we used a mixed strategy to approximate each node's centrality, called mixed centrality, $M(v)$. For a graph G, we will calculate each node's degree centrality and closeness centrality, and then get the degree centrality rank ($r_1(v), 1 \leq r_1(v) \leq n$) as well as closeness centrality rank ($r_2(v), 1 \leq r_2(v) \leq n$) of each node. Then the mixed centrality is computed as the linear combination of these two ranks:

  $$M(v) = 0.5(n - r_1(v)) + 0.5(n - r_2(v)).$$

  The running time of computing $M(v)$ is almost the same as $C(v)$, since computing $C(v)$ is dominant.

  We found that this mixed centrality performed better than pure degree or closeness centrality, so finally we used the mixed centrality to measure the importance of each node in a graph.

5 Game Theoretical Principles

- **Motivation**

  Concerning the rule that if more than one teams choose the same node, they will cancel each other, we started to think about strategies related to game theory. Let's draw a simple example of choosing one node (which has high priority in our centrality measure) in competing against TA teams:

  Table 1: Game Theory Example

	TA chooses this node	TA doesn't choose this node
We choose this node	Neither one gets this node	We get this node
We don't choose this node	TA gets this node	Neither one get this node

  From the table we can see that apart from accurate centrality measures and clustering methods, we also need to avoid collisions with other teams, especially when competing against multiple rivals. Another vivid example is shown in the following:

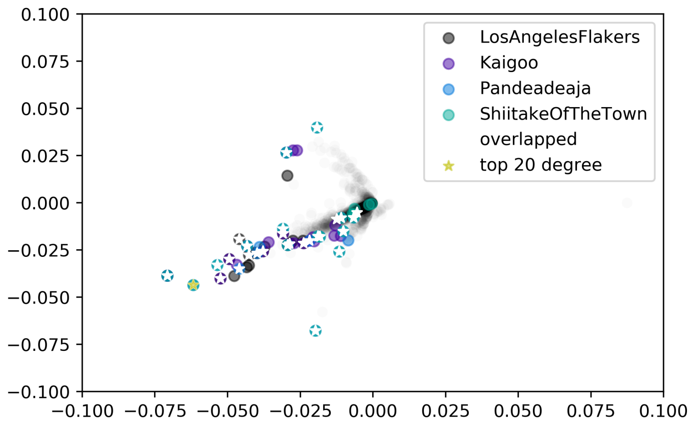

Figure 5: Laplacian spectral distribution of Graph 8.20.5.

The circles with white star inside correspond to nodes shared by multiple teams, and we can see there is a huge amount of collisions. Frankly speaking, **collision avoidance is a key part in designing algorithms**, especially when graph is highly centered on limited number of nodes.

- **Strategies**

  i. Our first strategy is simply choosing nodes with highest centrality measure, that is to say, we repeatedly submitted our nodes for 50 times, in other words "constant strategy". It did not work well and we often saw our team getting zero nodes in early days. The reason is that some other teams were also using "constant strategy" and we kept canceling each other in 50 rounds.

  ii. Our second strategy is picking more nodes than required, say, 1.5 times of number of nodes to form a larger set, and then randomly sample nodes from this set. We noticed that there is a tradeoff: the more centered nodes we choose, the more likely we can infect other nodes, but we are more prone to collision. This strategy worked better than the previous one, but later on we found it hard to adjust the time factor, and this method only decreased the number of collision nodes, but didn't really avoid collisions since we are still choosing from top nodes.

  iii. We then turned to a tricky strategy. Before we pick nodes, we throw out the top nodes according to our measure, say, top 1% of all nodes in the graph from our selection set. What's more, we tried the Mixed centrality stated above, in order to avoid the collisions with team using only one centrality measures. This strategy worked best in our approaches.

  iv. Apart from nodes-picking strategy, we also tried collision avoidance in clustering methods. We tried to deliberately pick the cluster with less nodes after a single cut of the graph. We found this strategy very unstable since we might pick some completely useless cluster. As a result, the combination of iii. and iv. helped our team reached the third rank in the second day of tournament. But in the last day, it did not perform very well.

6 Strategy Summary

- **Strategy to beat TA**

i. To beat "TA-fewer", we used the simple top N closeness centrality strategy, which means we just chose N nodes with largest centrality strategy.

ii. To beat "TA-more", we used random top N closeness centrality strategy, which means we randomly chose N nodes from xN nodes with largest closeness centrality, where x is a constant bigger than 1, and $x = 1.5$ in this case.

iii. To beat "TA-degree", we used **"random crack"** strategy. In this case, actually we know TA's node list, so we generated random node lists from xN nodes with largest closeness centrality ($x = 1.5$), and found one set of nodes that won against TA's choice. Then we used this set of nodes in all the 50 rounds.

- **Strategy in multi-player games**

  We analyzed the strategies of teams of top ranking with Laplacian spectal distributions. There two teams that we pay much attention to: **LosAngelesFlakers** and **Pandeadeaja**. We found they have completely different strategies.

  – **LosAngelesFlakers** tends to occupy the dominant cluster. They select nodes only in the dominant cluster to increase the chances of taking the majority. Takes the example of graph 8.10.5, they only select seeds on the left branch. And they win the game finally.

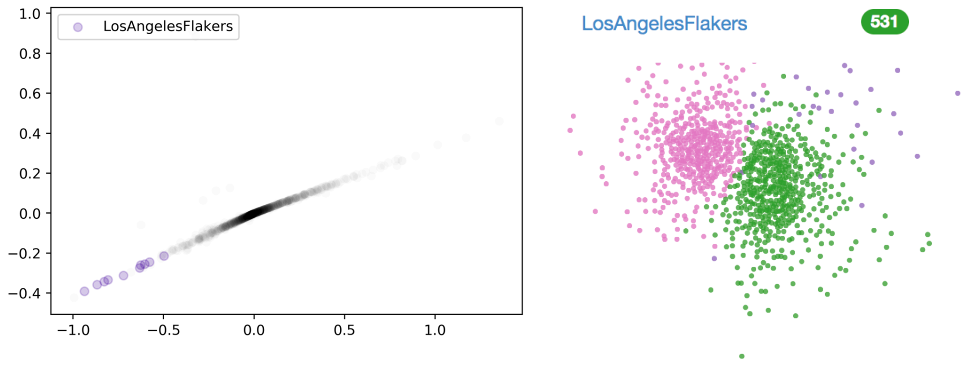

Figure 6: strategy of LosAngelesFlakers

  However, this strategy is very aggressive and sometimes risky since the dominant cluster has more competitors, so this strategy will either rank top or very low in the games.

  – On the contrary,**Pandeadeaja** takes s different strategy by trying to balance seeds among clusters, always pay reasonable attention to small clusters to get a more steady ranking. Taking the example of 27.10.2, we can notice the obvious strategy difference between LosAngelesFlakers and Pandeadeaja.

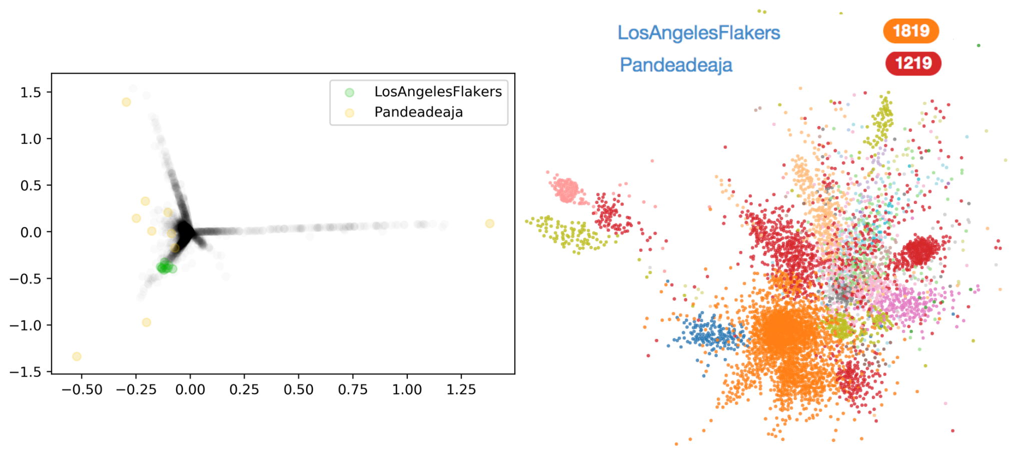

Figure 7: strategies comparison of LosAngelesFlakers vs. Pandeadeaja

For both of them, they all follows the rule of selecting nodes away from (0,0), especially Pandeadeaja, since nodes around (0,0) are less significant in the network spreading model.

We have analyzed above that in a multi-player game, the collision with other teams is top consideration, and in the end, **Pandeadeaja** with the balance strategy wins finally since it efficiently avoided overlapping by selecting important nodes in small clusters where most other groups tends to ignore.

Due to time limit, we didn't get a chance to formulate and test out a complete strategy with this spectrum properties. However, with some more experiences, a good strategy can be formed based on this spectrum distribution since it can most efficiently ignore the non-significant nodes, and reflect the importance and structure information of each node.

7 Suggestions for Course Improvement

i. The length of this project is too long, which made all of us exhausted, especially after the final round of tournament on Saturday night. We think three normal days is enough for us to beat TAs; if not, two submissions each day is a good choice.

ii. The node canceling rule made us upset. We admit that this rule is necessary for competition, but we feel really bad when we found that most of our nodes are canceled by other teams. We tried our best to avoid collision, but we can't predict the future. We can't think of a better way, but we do think this rule makes the competition more random, and further from graph theory.

References

[1] Dijkstra's algorithm - Wikipedia
`https://en.wikipedia.org/wiki/Dijkstra_algorithm`

[2] Horaud, R. P. "A short tutorial on graph Laplacians, Laplacian embedding, and spectral clustering." (2012). `https://csustan.csustan.edu/ tom/Clustering/GraphLaplacian-tutorial.pdf`

[3] Louvain Modularity - Wikipedia
https://en.wikipedia.org/wiki/Louvain_Modularity

[4] Centrality Measures Based on Current Flow. Ulrik Brandes and Daniel Fleischer, Proc. 22nd Symp. Theoretical Aspects of Computer Science. LNCS 3404, pp. 533-544. Springer-Verlag, 2005.
http://algo.uni-konstanz.de/publications/bf-cmbcf-05.pdf