**UCSD Social Network Analysis**

**Thomas Hepner, October 11th 2016**

1. **Overview:** An analysis of UCSD social network data on Facebook.
2. **Data:**The data is a snapshot of a social network at UC San Diego consisting of 783 nodes, representing individuals, and 3784 edges, representing the friendships between individuals.
3. **Questions**:
4. **Minimum Dominating Set:** The minimum dominating set of a graph is the minimum subset of nodes such that every node not in the subset is adjacent to at least one node in the subset. In this context, the minimum dominating set is the smallest group of individuals in the community who have at least one relationship with every other member of the community.
5. **Detecting Communities:** UCSD is a large community of students and faculty. However, within this larger group are smaller sub-communities: business majors, football players, professors, etc. I want to find these groups within the larger UCSD community.
6. **Algorithms, Data Structures, and Answer to your Question**:
7. **Minimum Dominating Set:** The greedy algorithm was used to approximate the minimum dominating set of nodes. The algorithm utilizes a **HashSet** to store nodes, and a **Set** to save the dominant nodes. The approximated running time for the algorithm is 1 + log |V| where V is the number of nodes, the actual runtime for my dataset was 0.0082 seconds. The algorithm returned a set of 266 nodesas the minimum dominating set out of783 nodes in the graph, or ~34%of all nodes in the data.
8. **Detecting Communities:** The Edge-Betweenness algorithm, using the Breadth-First Search (BFS) algorithm for each node in the graph, was used to detect communities in the social network. The algorithm computed the betweenness of all edges in the graph, and then removed edges with the highest betweenness until a user-specified number of communities was created. A **List** was used to store the subgraphs representing smaller communities within the social network, a **Map** was employed to record the shortest paths from a starting node to a goal node, and a **Stack** was utilized to distribute flow along the edges in the shortest paths. The algorithm returned a list of 267 communities in the larger social network; the community size of 267 was decided upon because it was the number of nodes in the minimum dominating set. One of the 267 communities contained 336 nodes, nearly 43% of all the nodes in the data.
9. **Algorithm analysis – Detecting Communities:**



Detect communities in the social network:

- Compute betweenness of all edges

- For each node v: O( |V| ( |V| + |E| ))

- BFS of graph starting at v O( |V| + |E| )

- Compute # of shortest paths from v to each other node O( |V| + |E| )

- Distribute flow to edges along these paths O( |V| + |E| )

- Remove edge of highest betweenness

- Repeat until the graph has desired # of components O(n)

The theoretical running time for this algorithm is n ( V2 + |VE| ) as demonstrated in the table above. The actual running time of the algorithm for 267communities was 23 minutes.

1. **Correctness verification (i.e. testing):**

To verify the correctness of my algorithm, I designed 2 small test cases. The first test case had 2 communities connected by a single edge, and the second test case had 5 communities of nodes each connected to each other by a single edge. I expected the algorithm to cut the edges between each of these communities when the appropriate community size was set, in these cases the community sizes were 2 and 5 respectively. The tests were completed successfully in a fraction of a second so I knew that I could apply the algorithm to the larger Facebook dataset.

1. **Reflection**:

My original plan was to apply this algorithm to the entire UCSD Facebook data 14,948 nodes and 886, 442 edges. However, this dataset is 3,768 times larger than the dataset I used for this project, meaning that the runtime would be 4-8 times longer for every doubling in the dataset. This means that my algorithm would take between 48 and 241 days to complete. Obviously, given the running time of my algorithm, this was not a reasonable task to accomplish so I decided to go with a smaller dataset for this project.

**Code Overview**

For each class include a few sentences about the purpose of the class and how it helps store the data required

1. **Class name: CapGraph**

**Purpose and description of class:** The purpose of the class is to read and store the social network data as nodes and edges, and to run various algorithms on the graph. The class has 2 methods, addVertex and addEdge, to create the graph as it reads data from a text file. Other methods are algorithms and their helper methods including getEgoNet, getSCCs, getMinimumDominatingSet, and getCommunities.

1. **Class name: CapNode**

**Purpose and description of class: CapNode is a class that stores information about a specific node in the graph. It contains an integer specifying the CapNode, a list of CapEdges (next class described) to describe the other nodes the CapNode is adjacent to, and a double for storing the distance when computing shortest paths between nodes.**

1. **Class name: CapEdge**

**Purpose and description of class: CapEdge contains information about a connection between 2 specific CapNodes. It also stores a double respresenting the betweenness of a specific edge which is used for detecting communities in CapGraph.**

**Overall Design Justification:**

CapGraph is the main class of the program, with CapNode and CapEdge as supporting classes. CapNode and CapEdge were built to abstract away a lot of important processes of the program from the CapGraph class. CapNode and CapGraph are essential classes which store important information used to build the graph, and run various algorithms on it in the CapGraph class. The CapGraph class controls the building of the graph from the text file, implements specific algorithms, and runs the algorithms from the main method for simplification.