

Programming Project II, First Report

Illya Starikov, Claire Trebing, Timothy Ott

Due Date: April 22, 2016

1 Abstract

Social Networks have revolutionized the way we communicate, meet others, consume information, and essentially influence our day-to-day lives. To show Social Network's prominence, [here are the percentages of online adults who use social media](#):

Facebook : 71% Adults

Twitter : 23% Adults

Instagram : 26% Adults

Pinterest : 28% Adults

LinkedIn : 28% Adults

This is unprecedented. Seeing as an overwhelming majority of adults have some sort of social media account, this can be used to model real world relationships — through social graphs.

In this experiment we would like to examine the social graph through the follower-followee relationship.

2 Introduction and Motivation

As stated above, social networks play a dominant role in our lives. Through social graph's, we can examine real world relationships.

There are many reasons for this to be valuable, such as common interest, community detection, influence amongst followers, etc. For this experiment, we would just like to test on the following measures:

Degree Distribution Test the direct amount of followers compared to followees a person has.

Shortest Path Distribution Test the degree of separation between a follower-followee.

Graph Diameter Test the breadth of a community.

Closeness Centrality Test the dependence of a degree of separation on a singular person.

Betweenness Centrality Distribution Same as previous.

Community Detection Based on Closeness Centrality, find the diameter of a community.

This will give us a reasonable dataset for the relationship of a community in a social graph.

3 Proposed Solutions

4 Plan of Experiments

The major purpose of our experiment is to dissect and examine networks, so we propose the following plan of experiments:

1. Extrapolate data and store in a relevant data structure — in our case, a sorted `map<int, vector<pair<int, double>>>`
 - The `int` is the key to be used, signifying the origin.
 - The `vector<pair>` holds all the adjacent edges.
2. Iterate over the entirety of the data structure to determine degree distribution.
 - For *weighted and unweighted out degree*, it is simply counting the the `vector` size with respect to each key.
 - For *weighted and unweighted in degree*, making an efficient algorithm is still difficult — not only is by default $\mathcal{O}(n)$ but there is an efficient way of finding where the edges lead to. We accomplish this by a [Binary Search Tree](#). By iterating over every vertex and storing their destination in a binary search tree, we have achieved a sufficient algorithm.
3. Detect the shortest path via the [Floyd-Warshall algorithm](#) for directed graphs.
 - Make the graph undirected by making it symmetric about the $a_{i,i}$ elements $\forall i \in \text{edges}$.
 - Test again.
4. Detect closeness centrality and betweenness.
5. Implement community detection.
6. Output results to user.

5 Team Roles

- Illya Starikov
 - Project Manager
 - Official Write-up
- Timothy Ott
 - Pseudocode Write-up
 - Algorithm Analysis
- Claire Trebing
 - Pseudocode Write-up
 - Algorithm Analysis