Final Report for Lab 3

Group Members: Shiuli Das, Sanjana Wadhwa, Kevin Yen

Table of contents

Part 1: Exploring the Data

The Dataset:

The SNAP repository consists of datasets primarily collected with the aim of helping build a learning tool which can ease the maintenance of *ego-networks*, such as *lists* on Facebook, or *circles* on Google+. We chose the Facebook dataset from it because it was a substantially large data set, but at the same time, seemed manageable. This consists of data from 10 users, each of which have a unique id number associated with them. Namely, 0, 107, 348, 414, 686, 698, 1684, 1912, 3437, 3980. For each node (user) we have the data of edges in the ego-network, distribution of friends into different circles and features present for each user present in the friendlist.

Shortcomings of the data:

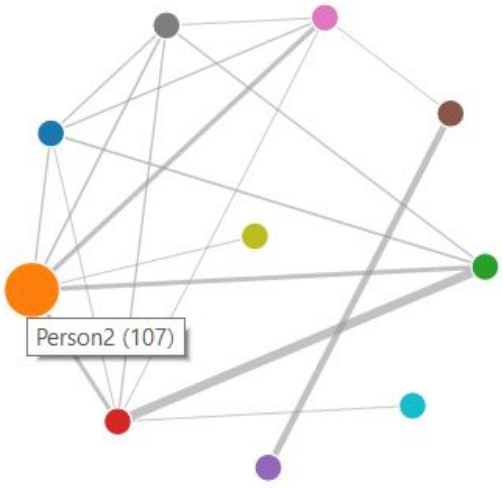
While preprocessing, we found discrepancies in the data, such as the node '346' was in 0.edges, but not in 0.circles, which should have been since all friends of node '0' should be in at least one of the circles. Our solution was to skip such values, because if they don't exist in the circles data they anyway won't help with any discovery.

Python Scripts:

(The following files have been attached for your reference, though the paths used are absolute.)

- Connections.py : Parses each *.circles* file and creates new files with all connections for each user as a txt file.
- Connect Edges.py : Uses the previously created txt files to create json file with nodes and edges for rendering Level 1 (description follows later).
- Level 2 Edges.py : Generates the data to generate graph of *.edges* file for each user.
- Level2EdgesMngfl.py : Generates the graph data for *.edges* file with weights as count of common features (more information in following sections).
- Data.py : Generated graph data for every pair of circles for each ego user node.

Level 1 : Connections of the users:



This represents the first level of the initial visualization. Each node represents a user, and the edge represents that there are common friends between any two selected people.
Applied visual encodings:

- Hover to highlight the node and get the node id (in brackets).
- Colours are random, since each node is a group in itself.
- Weight of the edge represent number of mutual friends.
- Hovering over the edge would give count of number of mutual friends.
- Fisheye was not needed on this level since the graph is not at all dense.

Level 2 : Closer Look at Each Person:

The interaction could enable you to click on each node to generate the Level 2 visualization. This encompasses the inter-connectivity of friends for each person. Each edge in the *.edges* file is represented by an edge in the graph.

1. This was a technique thought to get rid of any extravagant hairballs.
2. Forming such a hierarchical structure would have made data easier to comprehend.

Fisheye distortion was implemented on this level, because even for the smallest of friend lists, this level will be vulnerable to hairballs. The focus + context zooming can help concentrate on one particular area to see the interactions better.

In this graph, all edge weights are 1 simply if the edge exists. This lead to two major drawback of the design:

1. It didn't give any useful information.
2. One would expect nodes from the same group (belonging to one circle from the person's lists) (represented by same colour) to be cluttered (at least sort of) nearby.

We tried to solve this by making the weight of edge equal to number of common feature values of the two nodes. It would have the benefits of giving a more useful display of information, and decrease the size of generated hairball, as well. Since, intuitively, people in same circles should have common profile features.

Level2EdgesMngfl.py was written with this purpose. But it resulted in barely any edges added to the json file. That is, people in same circle of any ego-node, rarely had a common feature added in their profiles.
Since, we were not able achieve the full power of the visualization from Force-Directed Layouts, we switched to **Flare Visualization**.

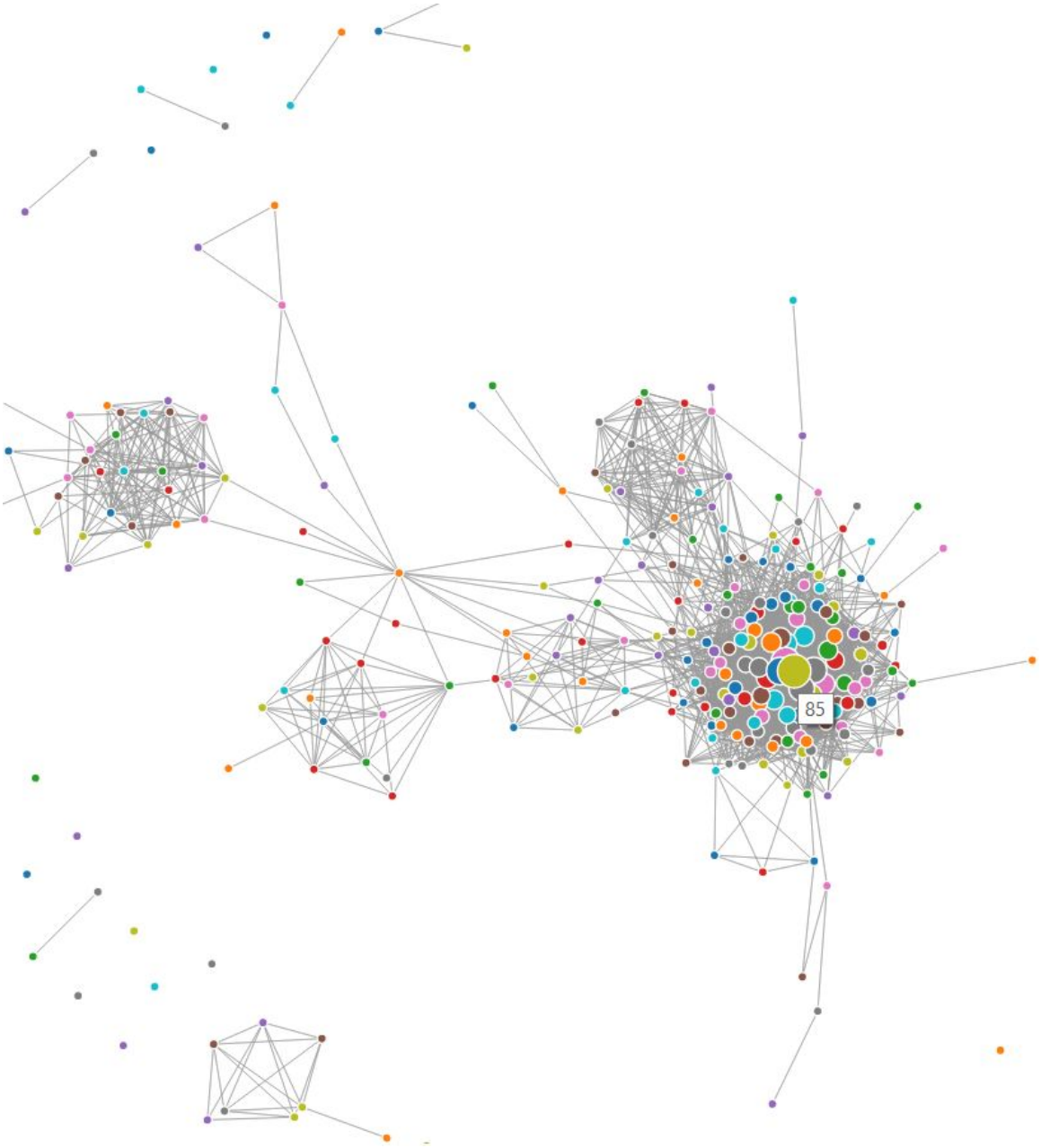


Figure: Sample from force-directed graph of 0.edges

Part 2: The Visualization

Overlapping Circles:

At times facebook users will add the same person in their friend list to more than one circle. The following visualization captures this.

Visual encoding:

- Central large node representing Ego Node (blue)
- Nodes in different circles have different colour
  (lower index: light blue, higher index: orange)
- Nodes present in both circles have same colour as Ego Node
- Hovering on the node reveals the node number associated with it

In the following figure we see that for Ego Node User 0, Circles 0 and 11 have 3 persons overlapping. The overlapping nodes among circles have the same colour as the ego node. One can hover on to the nodes to see the node number.

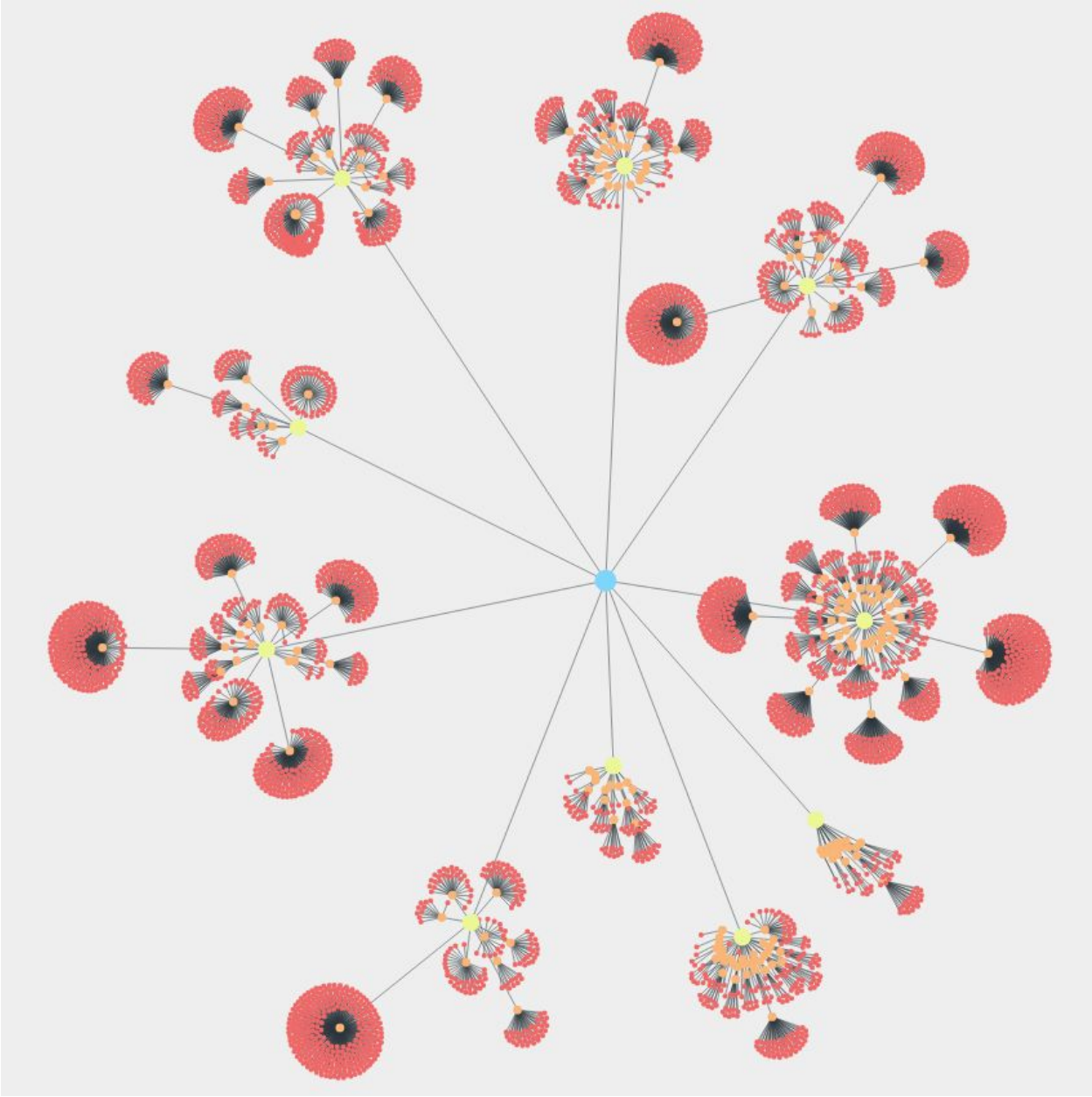
Fig: Ego Node: 0, Circles: 0 (blue),11 (orange)

In the following figure we see that there are none in common for Circles 0 and 5 for Ego Node 698. Also, we observe that each of these circles are very strongly interconnected.

Fig: Ego Node: 698, Circles: 0(blue),5 (orange)

This visualization helps us to focus on each circle of each ego user node without having to deal with clutter issues and make conclusions about overlap between circles and strength of connections within circles.

Flare Visualization:

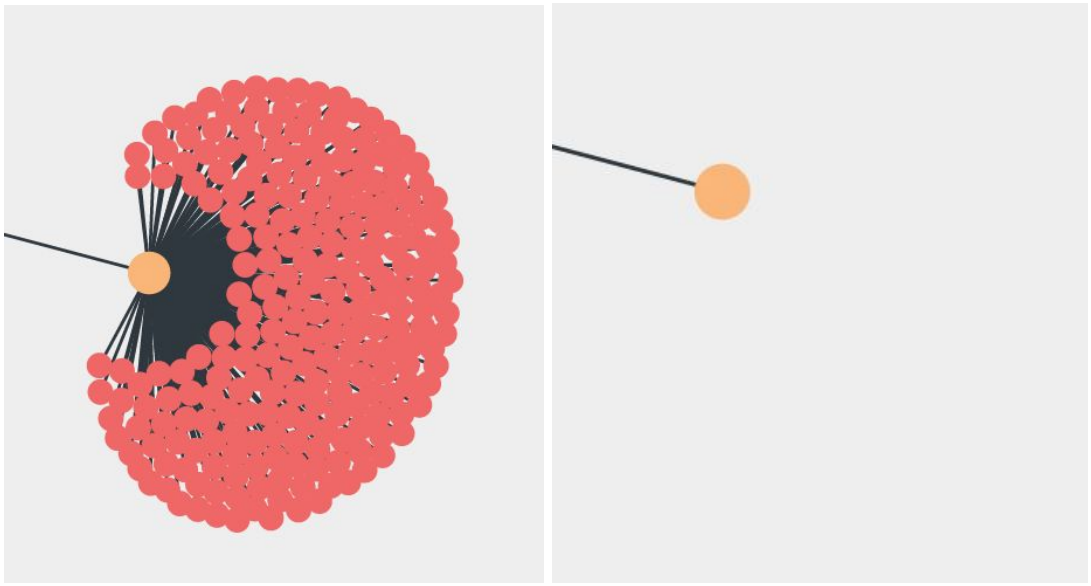


Overview

The **Flare layout** suits our visualization well, as it classifies nodes into different levels. To emphasize the level difference, we've used a color scheme of 4 colors. **Blue** represents an artificial node, named "FB", connecting to every ego nodes. **Yellow** represents *ego nodes*, in the dataset there are 10 people, thus there are 10 yellow nodes. **Orange** represents the *friend circle* of those 10 people. For example, it can be high school friends, college friends, or friends from class 5544. **Red** represents the *people* inside the friend circle. Above the the overview where all the nodes of this Flare graph are expanded. We've implemented some helpful features that allow user to better observe this graph, including [1]collapse/expand while clicking a node, [2]zoom and pan, [3]pause the force balancing, [4]tags on and off, and [5]fisheye distortion.

1. Collapse/expand

Like the example given from professor, our Flare graph allows user to expand or collapse each nodes if they have children. This process of expanding and collapsing is reversible and no information will be lost during the process. Below are the demonstrations of expanded node and collapsed orange node. This feature is triggered by user's mouse click.



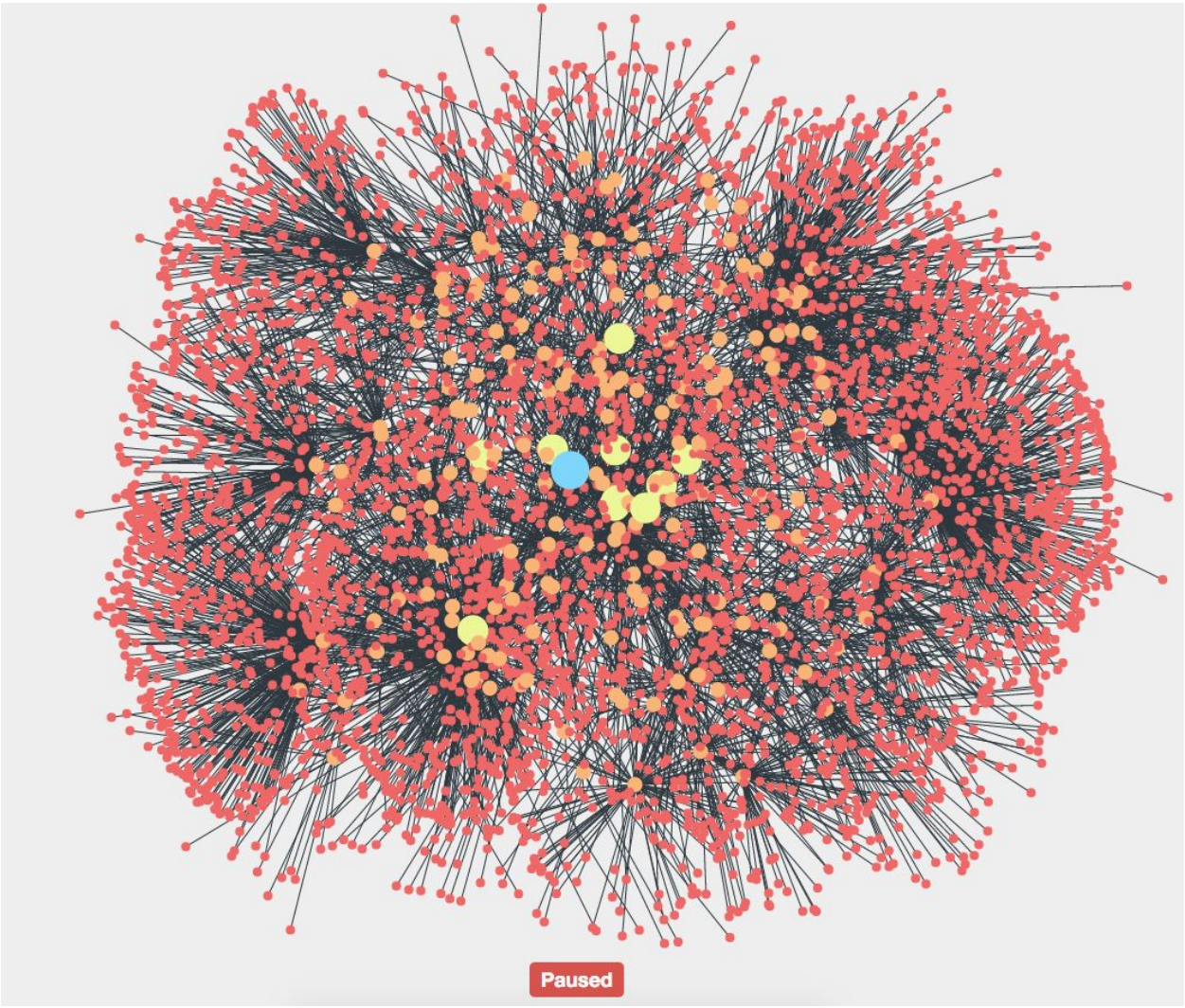
2. Zoom and pan

The ability of zoom in and move the graph around seems to be a natural instinct when we were given a complicated graph like our Flare graph, so we've implemented it. Be noted that the SVG canvas may not be able to hold all the nodes, especially when the user's screen is small, that's why the drawing area is mostly infinite, and user just have to control the camera

to see what he/she desires. This feature is triggered by mouse scroll and mouse drag and drop.

3. Pause

Force-directed graph is about balancing the force between nodes and edges, and it can take a very long time till it balances, while some may never balance. Therefore, the ability to pause the balance is very important if user wants to interact with the visualization, instead of watch it balance. This feature is triggered by keyboard press “P”.



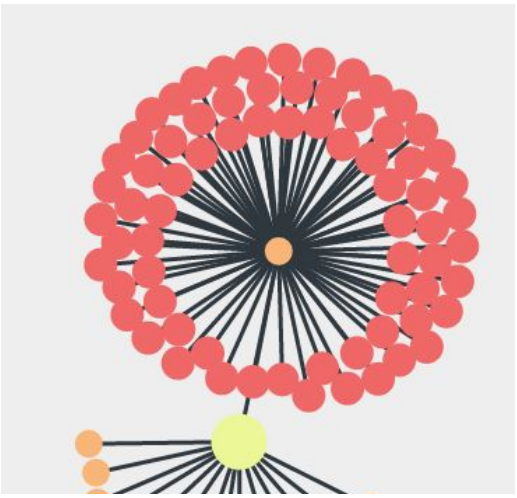
4. Fisheye distortion

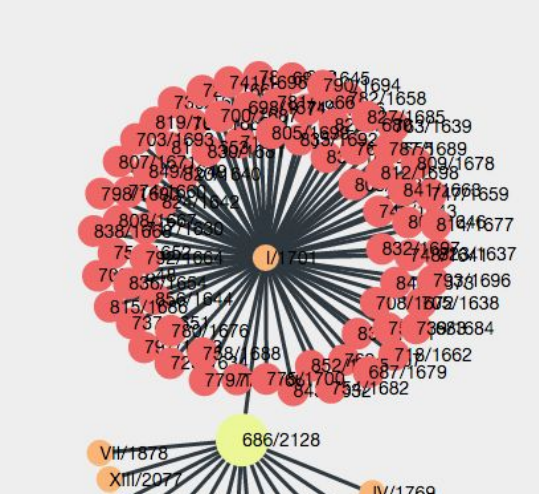
Fisheye allow user to discover more by distorting the objects within its range. We’ve implemented the basic fisheye, where objects are pushed away from the center of mouse

cursor. This feature is only available when the graph is paused, or stopped balancing. This feature is triggered by pausing the balance of force graph, which is keypress “P”.

5. Tags show/hide

There are data that come with each node, like their depth, or their node ID, and node name. In this case, we’re showing the node ID and node name. The reason for giving user the ability to show and hide the tags are for performance reason, also that enables user to have a more clear overview of the Flare graph. Moreover, showing the tags while balancing the force graph would be very slow, so we as well hide the tags, and show them only when we desire to see more information from the nodes. From the graphs below we can see that the tags are severely overlapped, due to the cluster of nodes, and that tags’ position are based on the nodes’ position. This feature is triggered by keypress “T”.





Shortcoming

The first shortcoming is that this Flare graph is not precise. One person can be a high school friend of A and B, thus this node should be connected to both circles. In this Flare graph, however, we’re not able to find these nodes that represent the same person and merge them into one single node. That’s why this graph is not precise.

Second, not enough features are implemented. We’ve planned to use this visualization to find mutual friends, but considering the time constraint, we were not able to finish it, and this feature should be based on the assumption that shortcoming 1 is resolved.