ILLINOIS INSTITUTE
OF TECHNOLOGY

ILLINOIS INSTITUTE OF TECHNOLOGY

# Project Report 1 - CS 595 - Fall 21

*Group :*

Ali Issaoui - A20474398

*Supervisor :*

Dr. Kai Shu

Sept, 2021

# Contents

# 1 Introduction

In this project, we will crawl social media data from Twitter, and then create and visual the Friendship Network of my twitter account. i.e. Represent all my friends on Twitter and whether there is a relation between any of them.

## 1.1 The choice of Twitter

At first, we wanted to create the friendship network of my Instagram account, since it is the social media that I use the most and where I have the most relations, however, due to the complexities in the Facebook API, the permissions that we need to have, and the privacy, we decided to go with twitter since the general method would be the same.

# 2 Data Crawling

## 2.1 library for accessing the Twitter API

In this project, the official twitter API for data crawling *Tweepy* was used. It is an open source Python package that gives you a very convenient way to access the Twitter API with Python.

## 2.2 The Twitter API

The first thing that we needed is to create a Twitter developer account, create an application and get 4 keys:

- The API key
- The API Secret
- The Access Token
- The Access Key

Then, we create and initalize the API as follows:

```
auth = tweepy.OAuthHandler(API_key, API_secret)
auth.set_access_token(access_token, access_token_secret)
api = tweepy.API(auth, wait_on_rate_limit=True)
```

At the end, the we will use the API to access and use the different functions available from Twitter.

## 2.3 The User

The first request that we needed from the twitter API is to look for the user that our project will be focused on ( me ). To do so, we used the function *get_user* as follows, using my account name *AliISSAOUI12*:

```
user = api.get_user(screen_name = 'AliISSAOUI12')
```

### 2.4 The Friends

The next step was to get the list of all my friends on twitter, i.e, all the people that I am following on twitter.

To do so, there are two possible ways: The first one is to use the Cursor object provided by Tweepy, which allows us to iterate through timelines, user lists, direct messages, etc

The second method is to use the function *friends_ids(user.screen_name)* provided by the twitter API, which returns the ids of all the friends of a given account by its screen name. For performance reasons, the second method was used in this project and creating the set of the friends of a given user was implemented as follows:

```
my_friends = set(api.friends_ids(user.screen_name))
```

## 3 Graph

### 3.1 Graph Data

The graph that is subject of this project is the Twitter friendships graph, so, the nodes in this graph will represent the twitter accounts who are my friends, and the edges will indicate whether the persons have connections with each other or not. The graph is oriented since in Twitter a person can follow another one without necessarily being followed back.

The data that will be needed to create this graph is the list of all my friends on twitter, and for each one of my friends, the list of the friends that we have in common (i.e. that we both follow ).

### 3.2 Graph Data Structure

For efficiency purposes, the data structure that was used in this project is a python dictionary. The graph is represented by a dictionary where the keys are the ids of my friends and the values are sets of the persons that we follow in common.

The set structure was used instead of a simple list since in Python, a set is considered as a Hashmap and the look up operation is very optimized.

Finally, given the list of all my friends that I obtained earlier, the dictionnary will be created as follows:

```
connections = {user.id: my_friends}
for friend in my_friends:
    connections[friend] = my_friends.intersection(set(api.friends_ids(friend)))
```

The first line initializes the dictionary with my twitter account as the first element, and the loop gets the set of the friends of every friend on my list and stores the intersection between this list and the set of my friends in the dictionary. The intersection means that those friends are friends in common between me and the actual friend that we are iterating on. Based on the Python documentation for *Set*, the complexity of the intersection of two sets is linear, so there isn't a major complexity problem.

However, the problem that we faced in this phase is that the twitter API only allows 15 requests of this type (e.g. get the list of friends ids), every 15min. So in order to operate on my list of around 200 friends, it took the program around 3 hours and 30 mins.

### 3.3 Saving the data

In order to avoid crawling the data each time we run the program, we saved the data using the pickle module for data sterilization. The entire connections dictionary is saved in the file named *"connections.p"* as follows:

```
with open('connections.p', 'wb') as fp:
    pickle.dump(connections, fp, protocol=pickle.HIGHEST_PROTOCOL)
```

And loaded as follows:

```
with open('../data/connections.p', 'rb') as fp:
    connections = pickle.load(fp)
```

### 3.4 Graph Visualization

Once we get the dictionary representation of the graph, we can visualize it using multiple packages. Two packages were used in this project.

#### 3.4.1 Networkx

The first package that was used is Networkx. Its use is very simple since we only have to create each edge and the plot the final graph. The implementation was as follows:

```
graph=networkx.DiGraph()
for person in connections.keys():
    for friend in connections[person]:
        graph.add_edge(person, friend)
nx.draw(graph)
plt.show()
```

The result that we got from this package is as shown in figure 1 and was not very visible and clear.

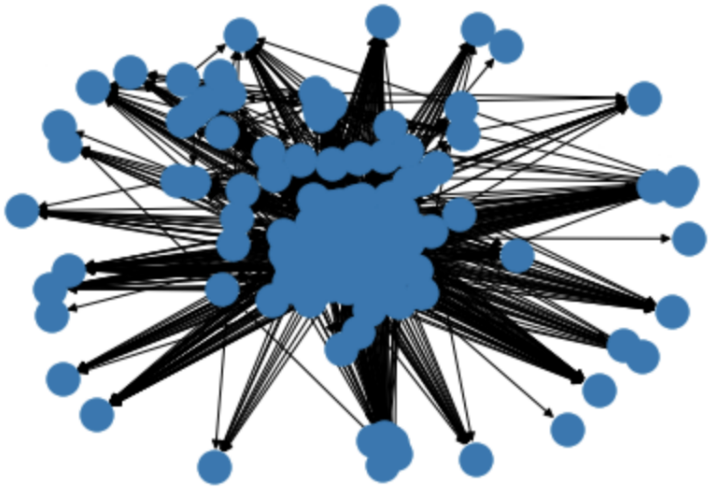

Figure 1: Graph with Networkx

### 3.4.2 Gephi

The second package that was used is *Gephi*, it is an open-source network analysis and visualization software package written in Java on the NetBeans platform. However, we needed to recreate the graph in format of a list of edges, i.e a list where each element is a couple of ids that represent an edge from the first node to the second node.

Gephi gives the possibility to visualize the graph using different algorithms, and colorize based on different metrics. The colors and the sizes of the nodes in pictures 2, 3, and 4 are proportional to the degree of each node. So the biggest nodes and those that are more blue are the ones with the highest degrees.

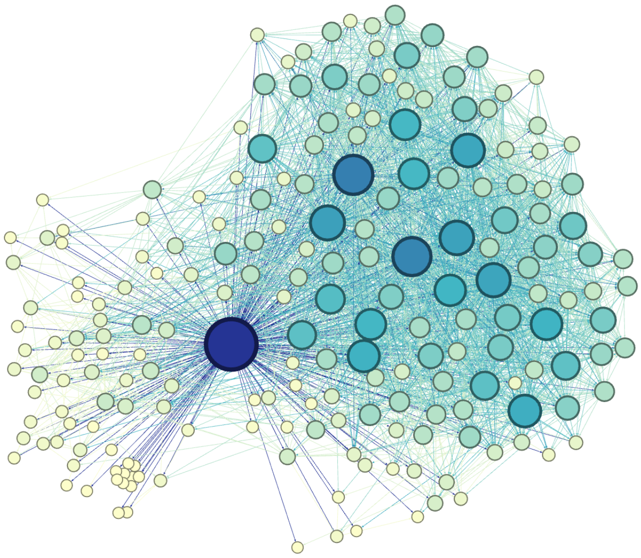

Figure 2: Graph with ForceAtlas layout

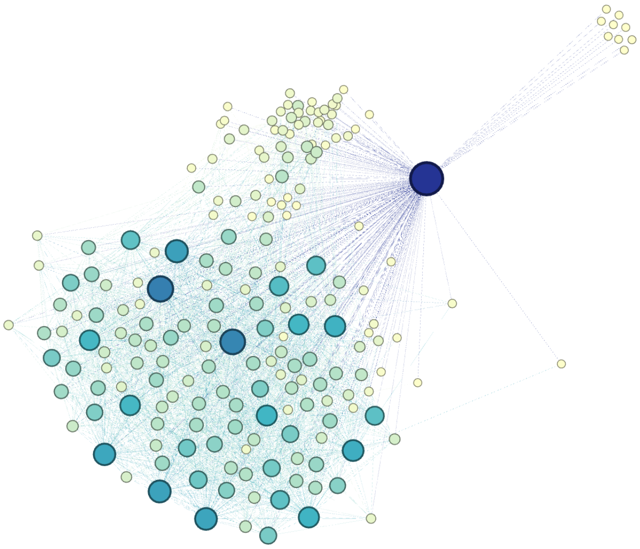

Figure 3: Graph with ForceAtlas 2 layout

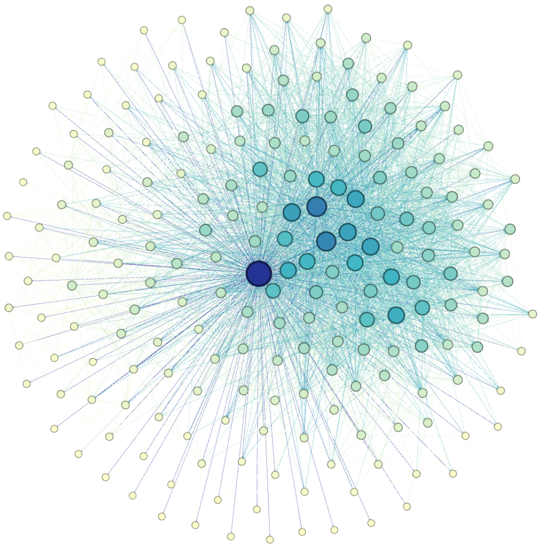

Figure 4: Graph with Networkx Fruchterman reingold layout

# 4 Graph Metrics

## 4.1 Number of Nodes and Edges

Our graph was composed of 204 nodes and 4919 connections.

## 4.2 Degree Distribution

In order to plot the Degree Distribution of the graph, we use the functions provided by the *Networkx* package. For example, give a node n, the function G.degree(n) returns the degree of the node n in the graph G. We use this function to create two lists of each degree and how many nodes have that degree, and we plot it to get the figure 5.

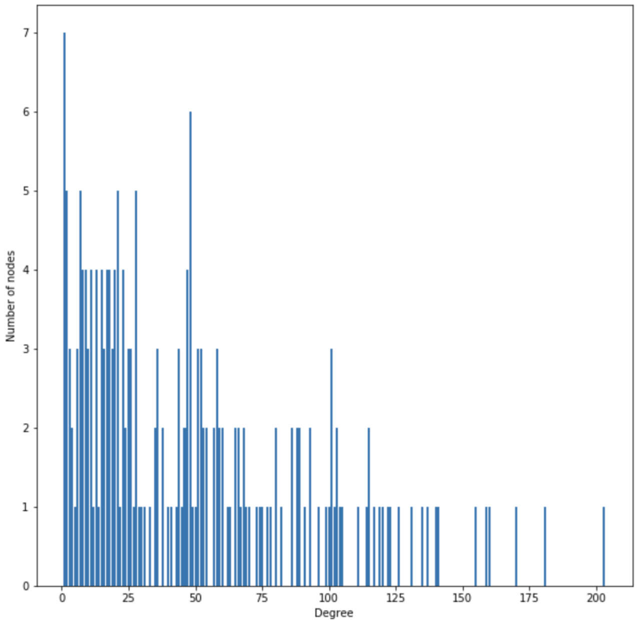

Figure 5: Degree Distribution plot

We can see that only a few nodes have a degree greater than 125, meaning that they are related to more than 125 other persons, the degree that was chosen in this situation combines both the in degree and out degree, i.e. a degree of 1 means that a person is following or is followed by one of my friends.

We can see that the plot is in accordance with the graph visualization, since very few nodes have big size and many nodes are smaller. The biggest node, the one with a degree of 203 is my Twitter account since it is connected to every other node of the network.

## 4.3 Clustering Coefficient

The second metric that we used is the clustering coefficient. This metric measures the degree to which nodes in the graph tend to cluster together. The plot of the clustering coefficient is shown in figure 6

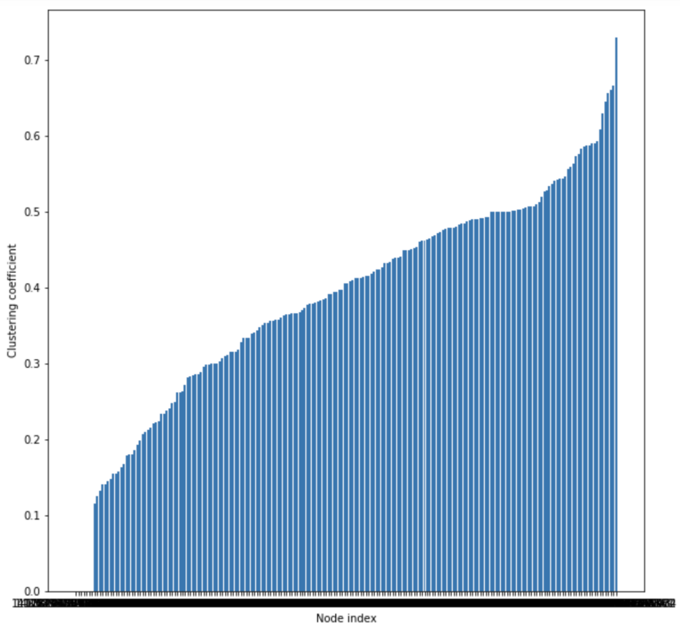

Figure 6: Clustering Coefficient

We can see that the clustering coefficient is very high for some nodes and relatively lower for some other, for example, the ones with the very low clustering coefficient are the users that I follow but don't follow me back and don't follow any one of my friend. Probably some people that are out of our society and surrounding. The nodes that have higher clustering coefficient are the friends that I follow and follow each other, probably a group of my friends who follow each other mutually, or a celebrity that everyone I know follows.

## 4.4 Betweenness Centrality

The betweenness centrality is a way of detecting the amount of influence a node has over the flow of information in a graph. It is calculated based on the number of shortest paths that pass through each node. We can see in figure 7 that the value of the betweenness centrality varies sharply from some nodes to others. It is maximal for my twitter account since it is in the center of the graph and all the shortest paths pass through it. It is also very high for some other nodes that are very popular locally, they can be some friends that I have in common with all my other friends, or some celebrities that are globally popular.

In the meantime, some nodes have very small Betweenness Centrality, these are the nodes that are isolated, for example the users that only I know and no one else of my friends knows, or probably some strangers that I don't know and have nothing to do with my surrounding but are still friends with me on Twitter.

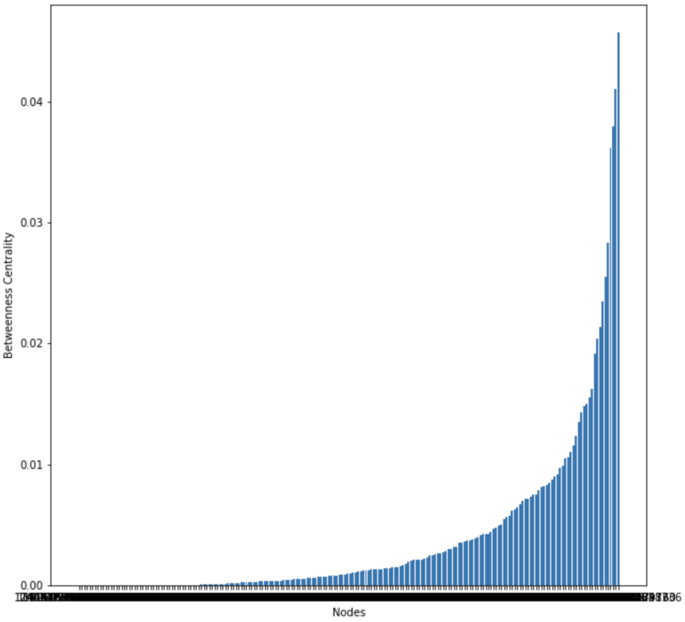

Figure 7: Betweenness Centrality

# 5 Conclusion

In this project, I have learned to crawl social data from a very popular social network, which is Twitter. I have also learned to choose the best data structure to represent a social graph and think about the potential complexity issues. Even though the graph has only about 200 nodes, it still gives an idea about the other complications that a biggest graph can have. This project was also an opportunity to know about the different graph visualization and analysis software and apply what we have learned in class to interpret the graphs based on the different metrics.