# Social Network Analysis For Computer Scientists

Assignment 2

Sevak Mardirosian
s2077086
s.mardirosian@umail.leidenuniv.nl

November 1, 2017

## 1 Introduction

This paper is relevant to SNACS course at Leiden University and specifically contains the answers for the second assignment.

### 1.1 Organization

In sections 1 and 2 I will try to answer the theoretical questions regarding Graphs. In section 3 more practical stuff are discussed and how to retrieve information from datasets using Python. I will, in few words, try to explain the ways I obtained my answers using which tools and techniques.

### 1.2 Tools and Techniques

For the programing parts I will be using a combination of Python 2.7 and Python 3.* scripting language with libraries such as **networkx**, **matplotlib**, **graph tool** and **pickle** which is already built-in both Python versions.

## 2 Clustering Coefficient

**Question 1: Name a) two types of graphs that have a clustering coefficient of 0 by definition and b) a type of graph hat has a clustering coefficient of 1 by definition**

The idea is as the following: A tree (graph), by definition, has no loops, and therefore has a clustering coefficient of 0. One other example as such is the bipartite graph. This type, at least of length 4, would only have cycles, simply because its impossible for two edges in the same partition to have a connection. Cycles of length 4 do not contribute to the clustering coefficient, because that only counts the number of triangles, i.e. the number of cycles of length 3.

**Question 2: Consider all possible connected undirected graphs with n = 9 nodes and m = 15 edges. Draw such a graph with a minimum average graph clustering coefficient.**

One can come up with different graphs. I drew mine below. Figure 1

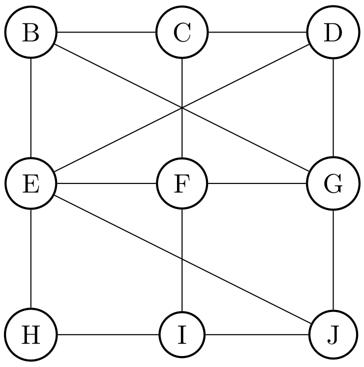

Figure 1: An undirected graph with 9 nodes, 15 edges, and a clustering coefficient of 0.

**Question 3: Give an algorithm, in words or in pseudo-code, for finding a connected undirected graph with a minimal average graph clustering coefficient for any given m and n. What can you say about the complexity of your algorithm?**

Creating an unclustered graph is pretty much streight forward when $m \leq n$ , this because one might draw a simple circle graph for instance (of size n) and afterwards, if required, remove the edges to get the desired number of edges. The algorithm I constructed works by connecting each node, lets say $v_i$ to antoher node $v_{i+\delta}$. The delta starts increasing by each iteration, and is given by $\delta = 3^s$ where $s$ is the current iteration number, starting at 1. The algorithm does the job of making sure no new loops are created shorter than 4 which is needed to prevent clustering. The algorithm below runs in $O(m)$.

**Algorithm 1** Undirected graph with a minimal average graph clustering coefficient

```
$E \leftarrow \emptyset$
$s \leftarrow 1$
while $|E| < m \wedge 3^s < n$ do
    for $v_i \in V \wedge |E| < m$ do
        $E \leftarrow E \cup \{\{v_i, v_{i+3^s \bmod n}\}\}$
    $s \leftarrow s + 1$
```

## 3 Exercise 2: Densest Subgraphs

**Use the greedy algorithm by Charicar et al to compute which nodes form the densest subgraph of the graph shown in Figure 1. Write down relevant state variables of the algorithm in each iteration as well as the different branching steps.**

Start by computing the average degree and the current density. For a graph given n,m this is quite straight forward. The average degree is $\frac{2m}{n}$ and the density is half of that. Running the algorithm would be by iteratively removing the nodes with a degree lower than the average degree from the $V$ and $E$. Repeat the process until we cannot delete nodes anymore. Using what I just described we end up with some pretty good results. Pretty much after the first iteration we already found the maximum with density of 1.5. Afterwards delete nodes twice until (to some point) you end up with a graph with only two nodes left and removing is not an option any longer.

Below the table of results.

| Iteration | Subgraph | Density | Avg. degree |
|---:|:---|---:|---:|
| 0 | $\{ABCDEFHIJKL\}$ | $\frac{16}{11} \approx 1.45$ | $\frac{32}{11} \approx 2.9$ |
| 1 | $\{ABEFJK\}$ | $\frac{9}{6} = 1.5$ | $\frac{18}{6} = 3$ |
| 2 | $\{BEFJ\}$ | $\frac{5}{4} = 1.25$ | $\frac{10}{4} = 2.5$ |
| 3 | $\{EF\}$ | $\frac{1}{2} = 0.5$ | $\frac{2}{2} = 1$ |

Table 1: Using the greedy algorithm to find the densest subgraph.

## 4 Twitter network analysis

**Question 3.1: Parsing the Twitter data.**

For the parsing the tweets I used a library called called `twitter-text-python`. One can easily use this library to extract useful information for stream of tweets. For starters I created a dictionary and a counter to count the tweets, afterwards I open the raw dataset file and start to strips things down. The library attempts to parse users (the owner of the weet) along with the mentions (people he mentioned) out of a tweet using a regular expression [built inside the library]. I went through the code and discovered that it uses a non-word character followed by an '@' followed by [1, 15] word characters, followed by by a non-word character. The library has functions such as getting the urls, tags, etc. And one of them, is getting the users of a particular tweet. Further, I loop or every and each tweet and filter out user-names and email addresses which seemed to occur frequently in the dataset. I save the user-name as a first-mention followed by the timestamps. I check whether the user is already in the set, if yes don't add those, if not add it by the end I return the results as an adjacency list. I then wrote another function to export all the number of users along with the first mentions to a csv file.
To use the dataset in the graph-tool library I wrote a helper function called create-edge-list which takes 2 arguments the adjacency list and the file in which you want to solve the results in. This handles the mapping from user-names to integers. It outputs a Gephi-edgelist compatible list of mentions to the standard output, and a Gephi-nodelist compatible user-name mapping to the standard error.
One of the things went wrong is how verify the determination of number of users. This because sometimes errors occurs when users have no separator (either white space or punctuation) between a mention and the following text which leads to lower number of nods (users) and this was quite confusing. To get some ideas I wrote another pure python tweet parser which resulted higher number of users. I was quite confused and frustrated. I asked the student assistent and he sugessted to grab just 50 users from the raw data and try to get the mentions of those. I did as such and got little over 26 mentions. I discovered that my parser was seeing the labels (Source, Target, Weight, Timestamp) as a node and edge. I deleted those and it returned 24 which was correct.
I decided to stick to the library given it has been tested by hundreds of users. But this could be solved using the Twitter api which provides functions such as users and other things such as timestamps and meta data. But given the time, I decided against this action and to deal of what

I have for now.

**Question 3.2: Present relevant statistics of your mention graph, including at least the number of nodes, number of edges, density, degree distribution and (approximated) distance distribution. What can you say about the size of the giant component?**

Continuing with the tweet parser I have build in the previous example we can use that to get some statistics about both datasets. Some of (sub)questions (number of nodes and edges, etc) we are already familiar using Networkx. For the Giant component however I have used the `weakly_connected_component_subgraphs` function in Networkx to calculate this. In both the large and small datasets things went smoothly and fast.
In general social network the average degree does not depend on the number of nodes in the network, we we expect the density to drop linearly with the number of nodes. Furthermore, I suspect the density will gradually decrease in the network and applies the same for giant component as well. All this because the network contains a lot of isolated nodes, i.e. people that have tweeted, but not mentioned, and who have not been mentioned.

The results are shown below.

| Dataset | $\|V\|$ | $\|E\|$ | $D$ | $\|V_{giant}\|$ | $\|E_{giant}\|$ | $D_{giant}$ |
|---|---|---|---|---|---|---|
| `twitter-small` | 65415 | 68336 | $1.6 \cdot 10^{-5}$ | 39443 | 52648 | $3.38 \cdot 10^{-5}$ |
| `twitter-large` | 648681 | 1020125 | $2.42 \cdot 10^{-6}$ | 496926 | 929233 | $3.76 \cdot 10^{-6}$ |

Table 2: Dataset statistics

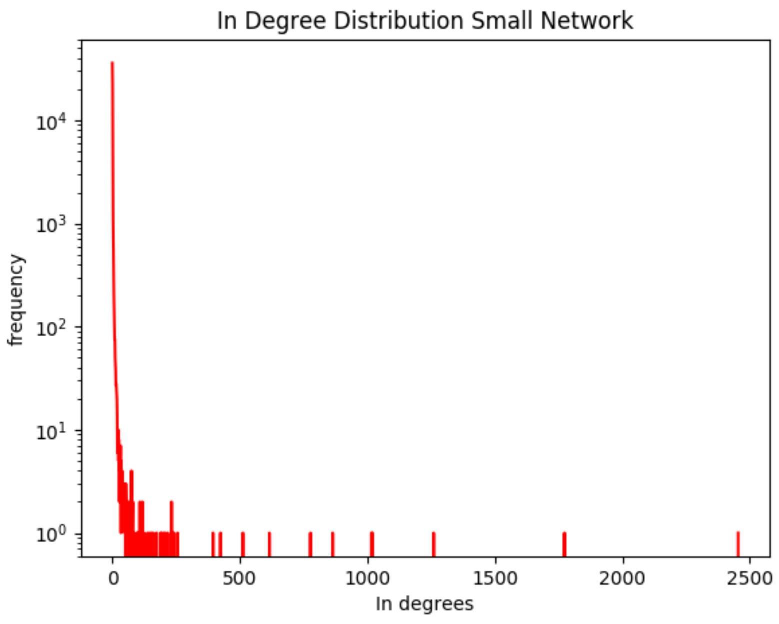

Figure 2: In-Degree distribution for twitter-Small.

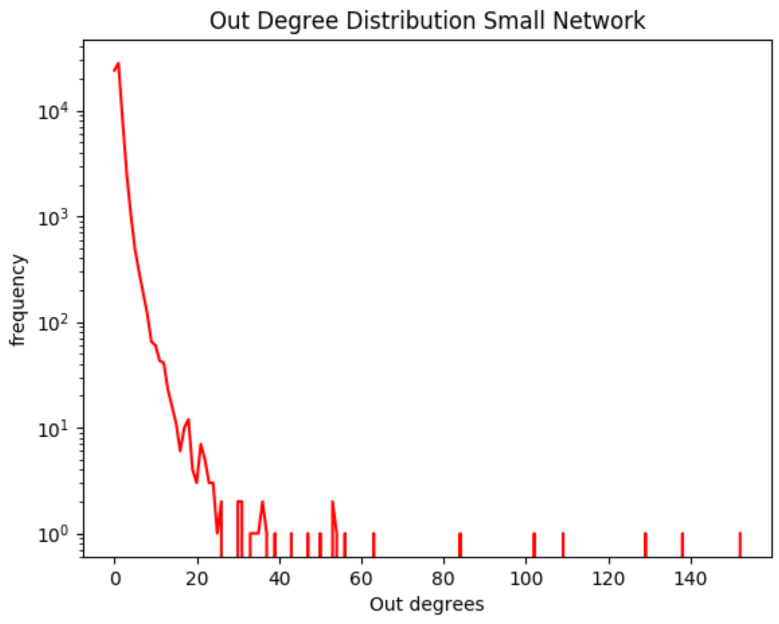

Figure 3: Out-Degree distribution for twitter-Small.

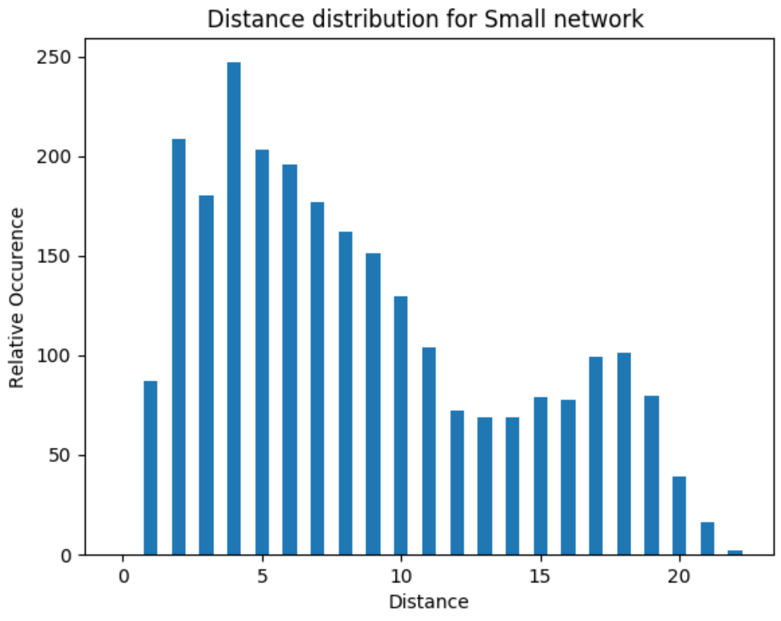

Figure 4: Approximate distance distributions for twitter-small.

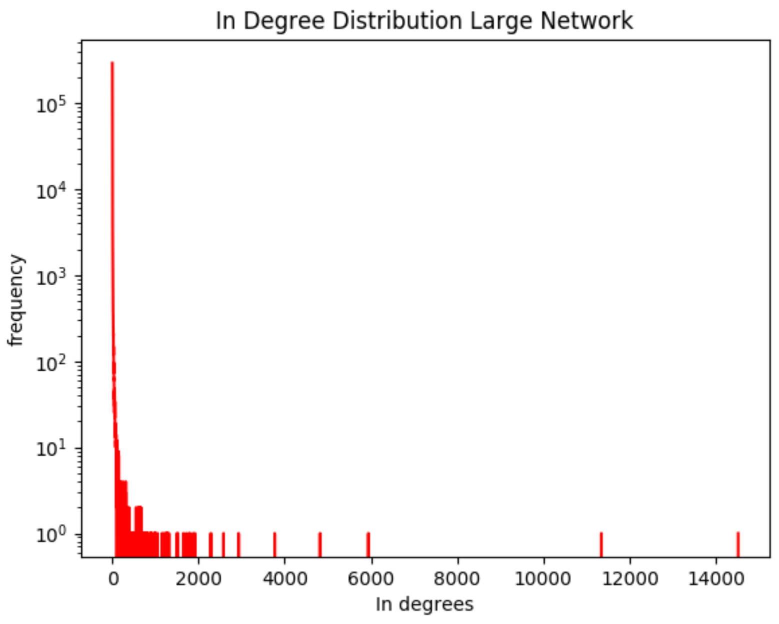

Figure 5: In-Degree distribution for twitter-Large.

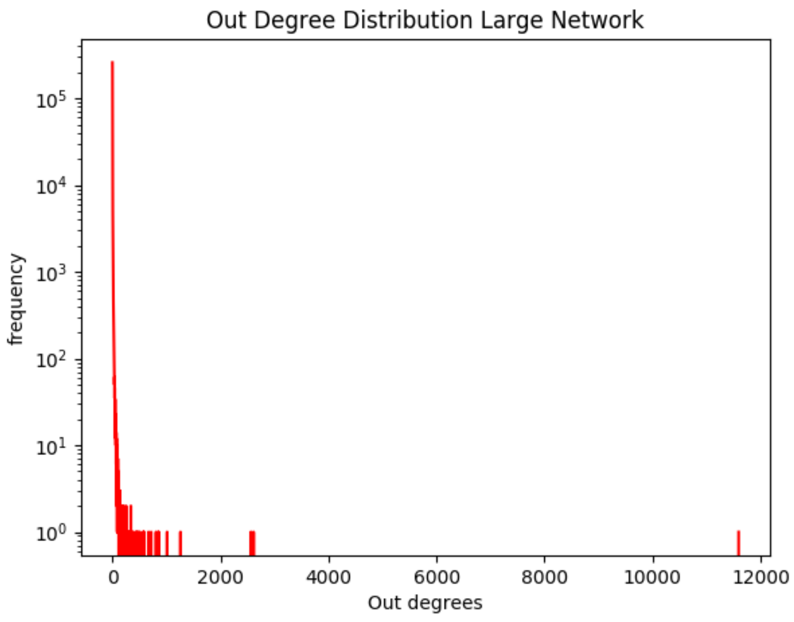

Figure 6: Out-Degree distribution for twitter-Large.

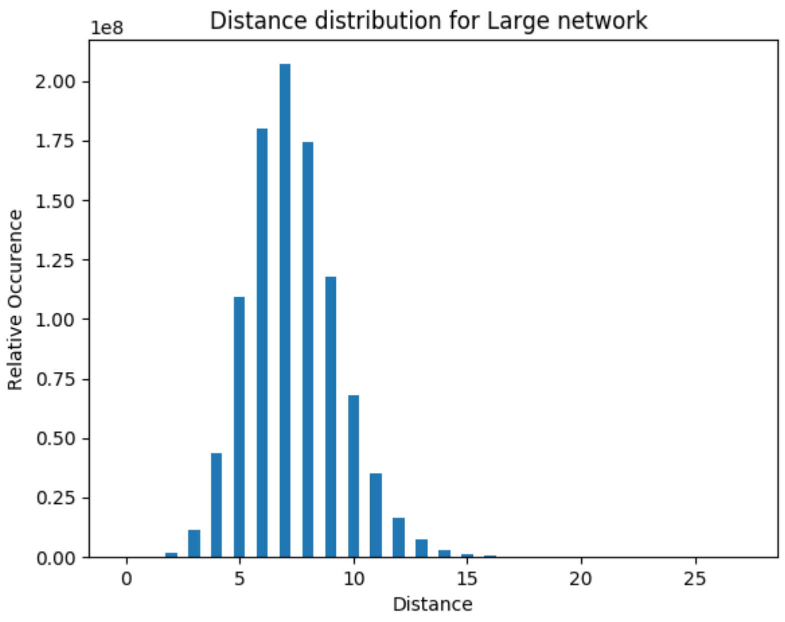

Figure 7: Approximate distance distributions for twitter-Large.

**Question 3.3: Determine the top 20 users based on three different centrality measures (for example, betweenness centrality, closeness centrality and degree centrality). Discuss the results. Can you come up with a way to numerically compare the similarity of the rankings?**

To compute the top 20 the following measurements were used.:

**Betweenness centrality** which is the longest shortest path starting from a node.

**Closeness Centrality** which is is somewhat related to betweenness centrality. In closeness you take the average path length rather than the total path length.

**Degree centrality** which is the degree of a node. For instance I've used the in- and out degree of a node for this. We then sort the nodes in descending order and grabbed the 20 top ones.

We can not easily and objectively compare these rankings on quality, but we can numerically compare how different they are. We do this by counting inversions. This can be done in $O(l^2)$, with $l$ the number of items in our list. This is fairly doable for a top 20. Also, we have to account for nodes that are in one, but not in the other list. For this, we count it as an inversion against everything else, which means it has $l$ inversions.

I applied the above measurements to the `twitter-small` dataset. Results are below.

| | $In - DC$ | $Out - DC$ | $BC$ | $CC$ |
|---|---|---|---|---|
| 1 | theiphoneblog | evansrobert | theiphoneblog | kitevc |
| 2 | ryanbarr | squaretrackle | filjedi | urkelbotbrian |
| 3 | mashable | igroaniphone | msproductions | bzwingzero |
| 4 | scottbourne | macandiphone | lemonkey | redproductions |
| 5 | ScanCafe | bingwaves | dreadpiratepj | stigblog |
| 6 | squarespace | russelsview | serban | maryg_pr |
| 7 | iphone_dev | iphonedef | iphonespaz | pierhousekw |
| 8 | tweetmeme | rtapple | webaddict | msdixon |
| 9 | TweetDeck | wootboot | johnbiggs | ulyzp |
| 10 | kevinrose | iphoneincanada | dgshaw | failtracker |
| 11 | TomTom | failbus | hownottowrite | stephsolo |
| 12 | TechCrunch | mac0s | corvida | giggyz |
| 13 | patrickaltoft | mizamandareed | techcrunch | budgibson |
| 14 | QuickPWN | top_iphone_apps | imeem | adoniramsides |
| 15 | engadget | allthingsiphone | reneritchie | sjusjun |
| 16 | iphoneincanada | talosman | iphoneincanada | zwiebertjuh |
| 17 | tinteract | luv_nokia_n97 | krystynchong | paulschwend |
| 18 | TUAW | andyoniphone | mayhemstudios | jaykahn |
| 19 | guardiantech | trackle | talosman | carlosblaze |
| 20 | MuscleNerd | razorianfly | razorianfly | stealthsolidus |

Table 3: Top 20 users in `twitter-small` according to different measures.

|  | *In − DC* | *Out − DC* | *BC* | *CC* |
|---|---|---|---|---|
| 1 | mashable | uberguineapig | uberguineapig | bknowles314 |
| 2 | taptaptap | xwordsaddict | razorianfly | smeology |
| 3 | TechCrunch | mobil_tipps | taptaptap | h3rry4nt0 |
| 4 | tweetmeme | iphonedevnews | mmartel | iappreviewer |
| 5 | theiphoneblog | iphone_mob | theiphoneblog | mrsdjlsd |
| 6 | iphone_dev | followermonitor | marcoarment | saam_saam |
| 7 | freeiphoneapps | iphone_jedi | appbank | ant_1 |
| 8 | tysiphonehelp | iphonebestacc | tommytrc | captmolson |
| 9 | ryanbarr | macandiphone | iphonegirl | makensyyy |
| 10 | TomTom | tm_iphone | simplytweet | cerri_andthebox |
| 11 | iphonefan | maclounge | adriarichards | dietplancoach |
| 12 | engadget | julie_warner | tweetmeme | jkblacker |
| 13 | TFLN | iphonejobsi | ikeafans | meghyyyy |
| 14 | TUAW | botiphone | iphone_dev | j3nna |
| 15 | Maggianos | vara411 | iphone_mob | dewiiweddeled |
| 16 | WeekinRewind | ipodtouchfanno1 | filjedi | dsgnbycassandra |
| 17 | parislemon | smartonlinetwit | iphonefan | nwinton |
| 18 | joehewitt | iphonefuns | mayhemstudios | bdosono |
| 19 | TweetDeck | iphone_nikki | dudeman718 | nayannababii |
| 20 | macTweeter | razorianfly | iphoneincanada | faberzinho |

Table 4: Top 20 users in `twitter-large` according to different measures.

As you can see some of the top users are in common in either In-DC and Out-DC or BC. This is somehow not unexpected, as all measurements used for determining the rankings are correlated.

**Question 3.4: Apply a community detection algorithm (such as Modularity in Gephi) to the giant component of your mention graph, and try to manually interpret and discuss the results. How did you set the resolution parameter?**
For this I have used gephi to get the job done. Ideally, I was looking for about 20 communities, as a lower amount would provide very little information and a higher amount would not be visually interpretable. When setting the resolution to 1.0, I got thousands of communities. When I set it to 10.0 only 4 communities came back. Halfway in between at 5.0 18 communities were viewed, which can be roughly interpreted as some subject on which users tweet.

**Question 3.5: Visualize the giant component of the network**
This was one of the challenging ones. Using Gephi is honestly not my strongest skills. I struggled, and I struggled a lot. I loaded the twitter-small (or at least I tried to load it) and things weren't comfortable in Gephi. At first it starts freezing, and crushes afterwards. I have tried to get it to work on three different computers just for this. Finally I have somehow managed loading the tweets (small) into the program. Using the same community detection as mentioned in the previous section I applied SF2 algorithm, let it stabilize for a moment and colored the nodes. I tried to improve it but as I said it kept crushing and wasn't having the best day. At some point I had to stop spending some much time on it. Below is my final results.

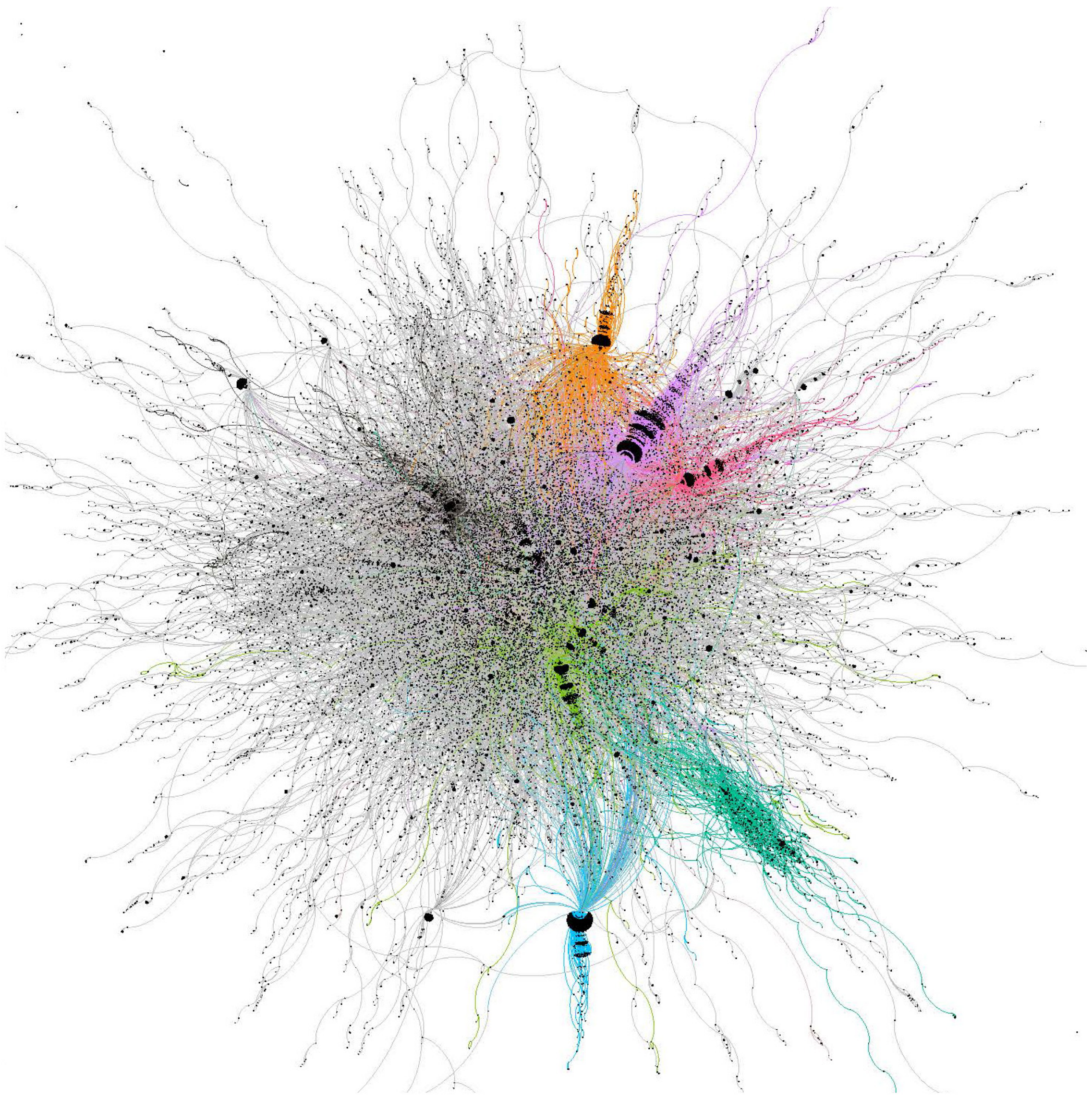

Figure 8: Visualization of the `twitter-small` network. Colors are according to communities, and node size is proportional to betweenness centrality.

**Question 3.6: Run your code on the larger dataset given in the file twitter-larger.in, and answer Question 3.2 and 3.3.**

Using the same parser as above I used this to retrieve statistics only this time on the `twitter-larger` dataset. To make things easier somehow I have used graph-tool[1] to get the information needed. For the results check 3.2.
Furthermore, Computing the betweenness centrality was the hardest part. It took quite some time to do compute this. Again I have used the betweenness centrality function in graph-tool to achieve faster results. It took me approx 2-hours to get some results. I wasn't planning to run this one more time. Getting the other centralities was achieved within 10-20 minutes.

[1]Graph-tool is written in C++ which I believe number of times faster than Networkx which is pure python library