**CSE 601 – FALL 2016**

**DATA MINING AND BIO INFORMATICS**

**HOMEWORK – 3**

**MARKOV CLUSTERING ALGORITHM**

**IMPLEMENTATION AND RESULTS REPORT**

Submitted by:

Arti Gupta

(50170010)

artigupt@buffalo.edu

Sudharchith Sonty (50169912)

[sudharch@buffalo.edu](mailto:sudharch@buffalo.edu)

**UNDERSTANDING:**

The **MCL algorithm** is short for the **Markov Cluster Algorithm**, a fast and scalable unsupervised cluster algorithm for graphs (also known as *networks*) based on simulation of (stochastic) flow in graphs.

The basic interface to the algorithm is very simple - we need only one option (the **-I** flag) to get to the heart of it, and for large graphs you should also be aware of the **-scheme** flag for regulating resources. The default approach is to vary the argument to **-I** over some interval (doing an MCL run for each value), and analyze the clustering output with the other programs that come with MCL.

There is no easy way to adapt the MCL algorithm to output a specified number of clusters. This is in my opinion, for 99.99% of the time a highly desirable feature. We would generate 4 or 5 clustering results at different levels of granularity (say setting the MCL inflation parameter to 1.4, 2.0, 3.0, 4.0 and 6.0, but it could be worthwhile to do a few more and pick based on the distribution of cluster sizes), then unify them in a hierarchical clustering After that one could traverse the tree and try to find an optimal clustering of the desired size. This obviously requires significant effort.

Considering a graph, there will be many links within a cluster, and fewer links between clusters. This means if you were to start at a node, and then randomly travel to a connected node, you’re more likely to stay within a cluster than travel between. This is what MCL (and several other clustering algorithms) is based on.

**IMPLEMENTATION:**

* We Have implemented the MCL Algorithm in python using a numpy 2D Array.
* For the first data set, we had directly generated a matrix with nodes from 0 to 179 as those values were already grouped and contiguous.
* For the generated Adjacency Matrix, we had followed the steps of the Algorithm as follows with a few additions.
* For the Yeast and the Physics data sets, the mapping was done as follows

1. We had put all the individual elements in a set and indexed it
2. Then we created a one to one mapping of each node to numbers from 0 to the maximum value in contiguous fashion.
3. Then, the matrix was created based on the mapping from the input file to the newly created map.

* **Example for physics set:**

BEN-AVRAHAM': 12, 'SREERAM': 119, 'OLTVAI': 92,

* **Example for yeast set:**

4302': 216, '3557': 177, '5824': 293

**ALGORITHM STEPS FOLLOWED:**

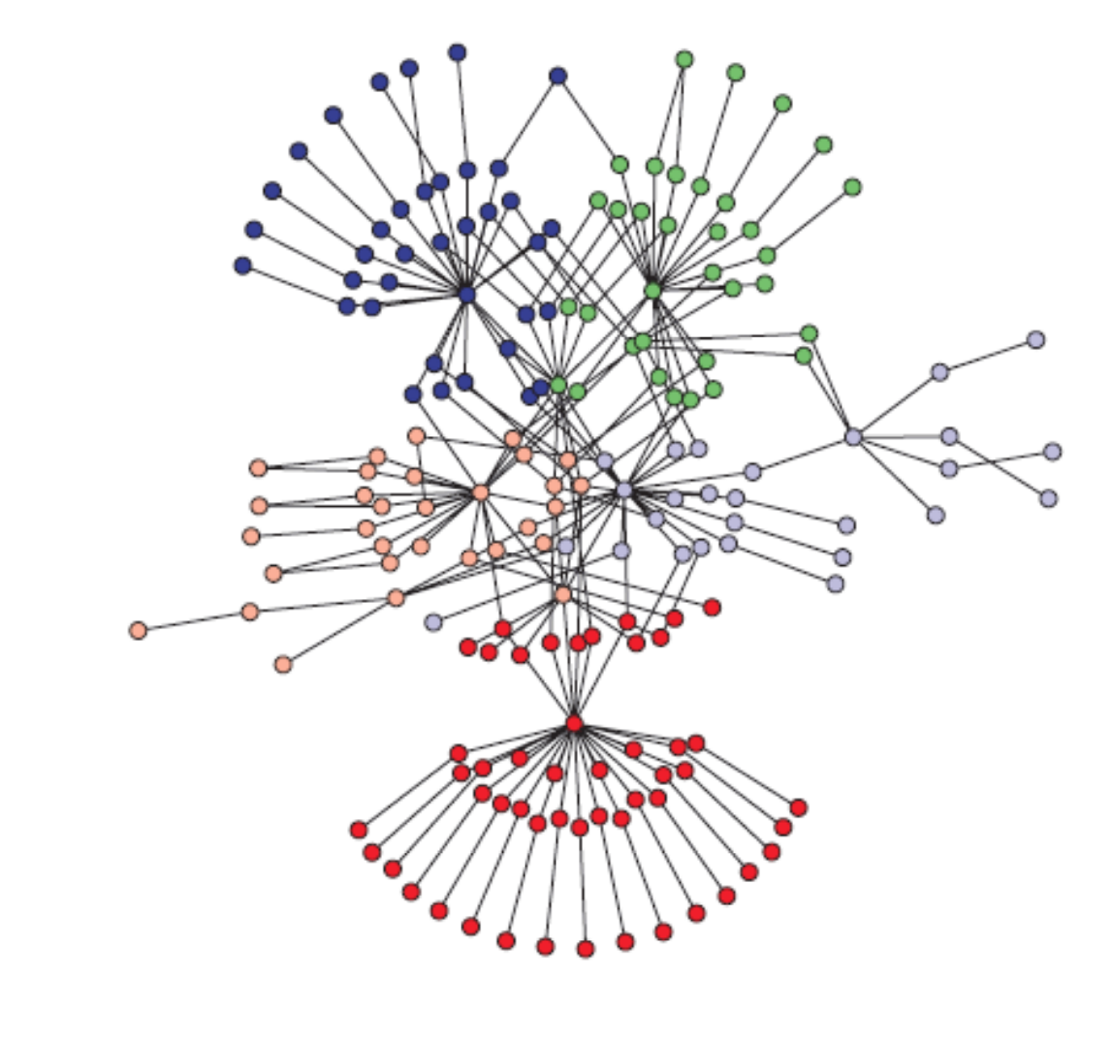
1. Input is an un-directed graph, power parameter e, and inflation parameter r.
2. Create the associated matrix
3. Add self loops to each node.
4. Normalize the matrix.
5. Round off the values in the matrix to two decimal points.
6. Expand by taking the eth power of the matrix.
7. Inflate by taking inflation of the resulting matrix with parameter r.
8. Round off the values in the matrix to two decimal points again after inflation
9. Repeat steps 5 and 6 until a steady state is reached (convergence).

* Convergence was checked on the basis of equality of the matrix from the previous state and the current matrix generated.

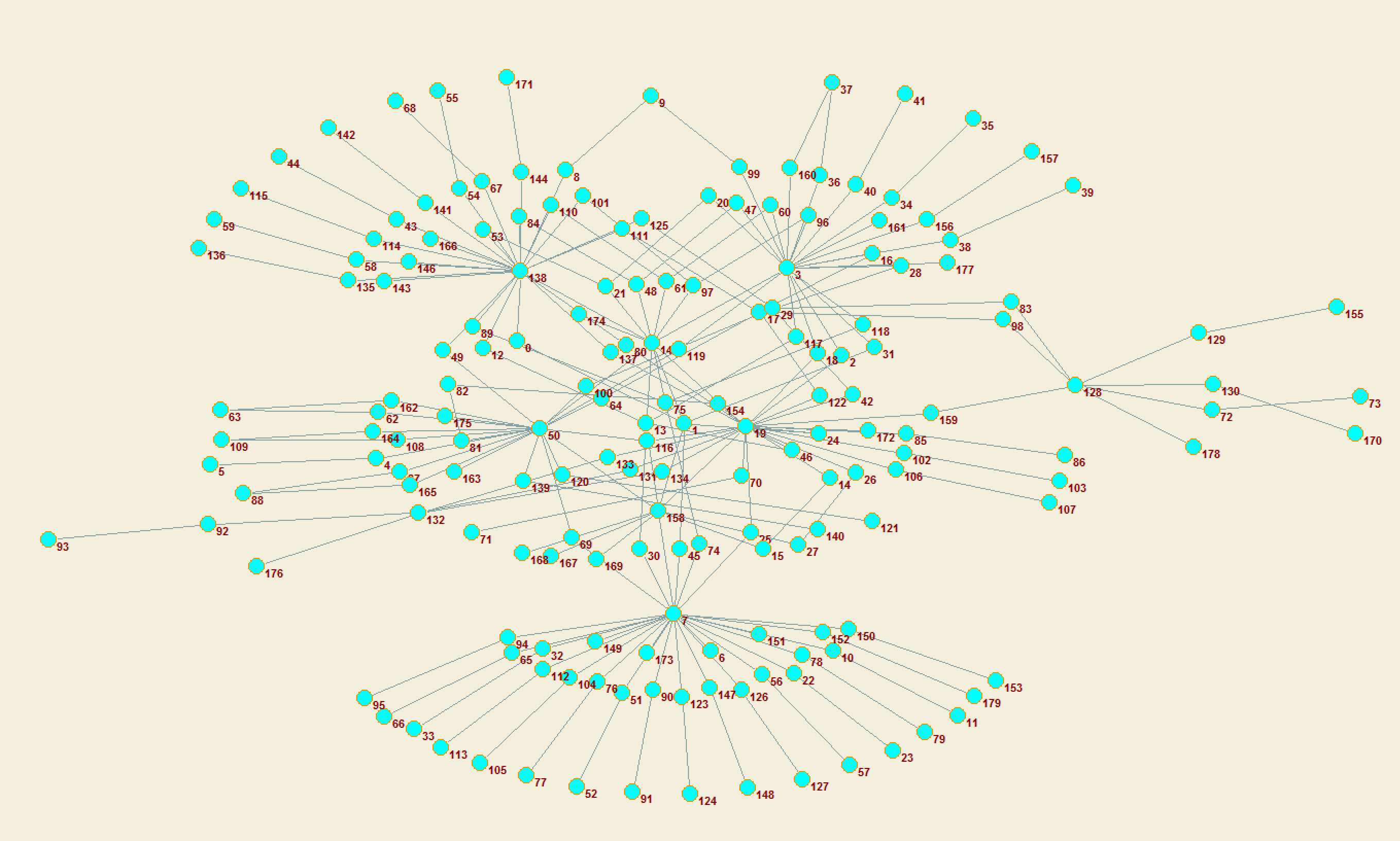
**RESULTS AND VISUALIZATION FOR DATASET 1:**

* Setting the Inflation parameter to 6 yielded the best steady state results and the results thus obtained are displayed below as follows.

**VISUALIZATION BASED ON GROUNDTRUTH:**



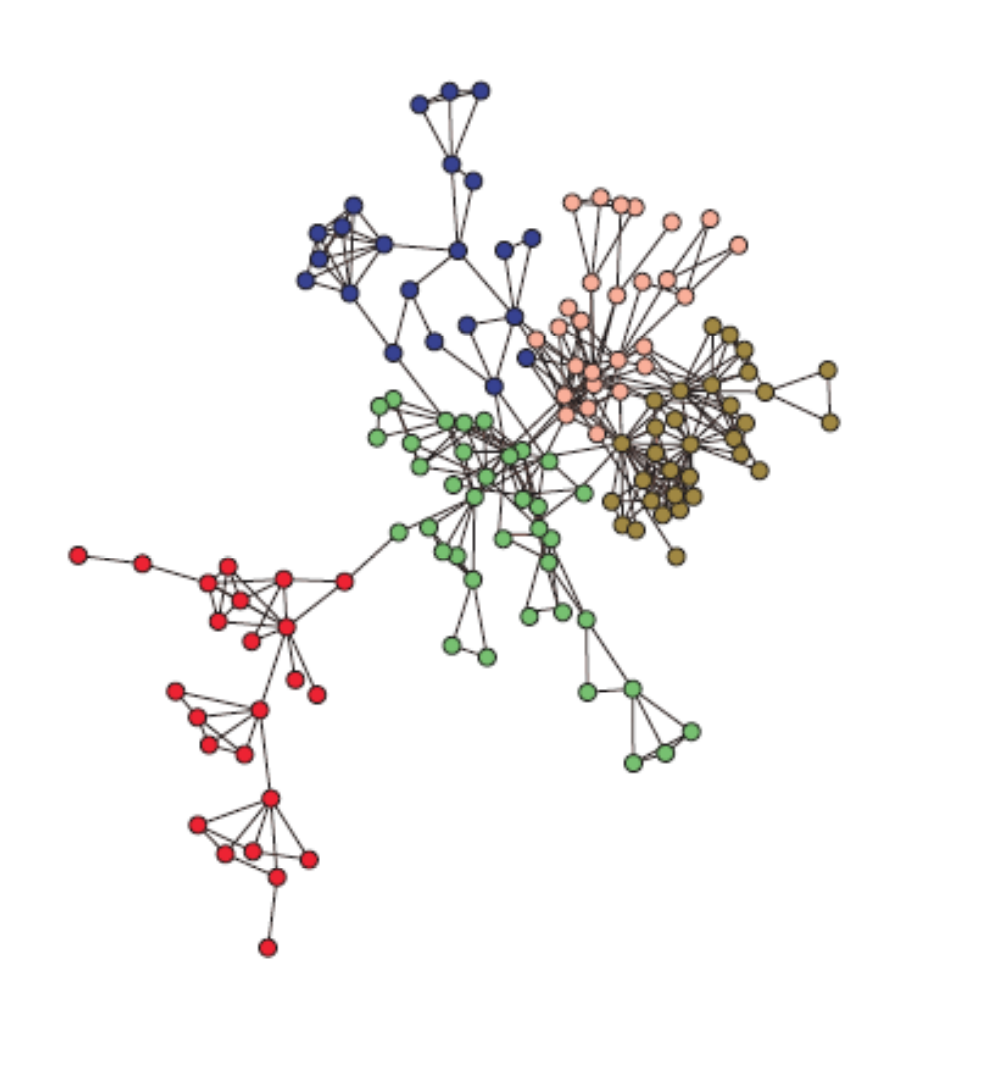
**VISUALIZATION OF RESULTS:**



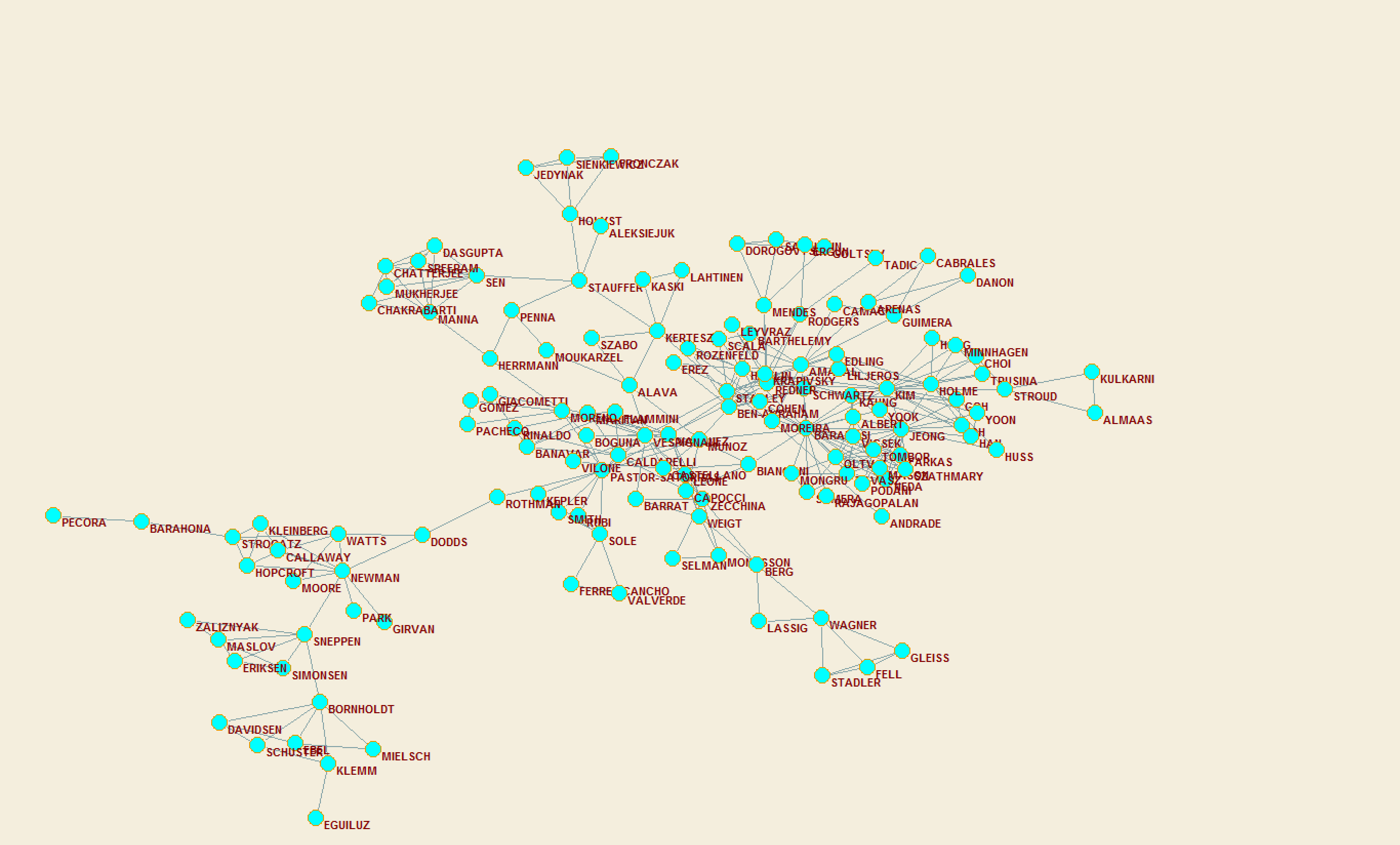
**RESULTS AND VISUALIZATION FOR DATASET 2:**

* Setting the expansion and inflation parameters to 10 yielded the best steady state results and the results thus obtained are displayed below as follows.
* Any Parameter setting after 10 seemed to have given more or less the same visualization.

**VISUALIZATION BASED ON GROUNDTRUTH:**



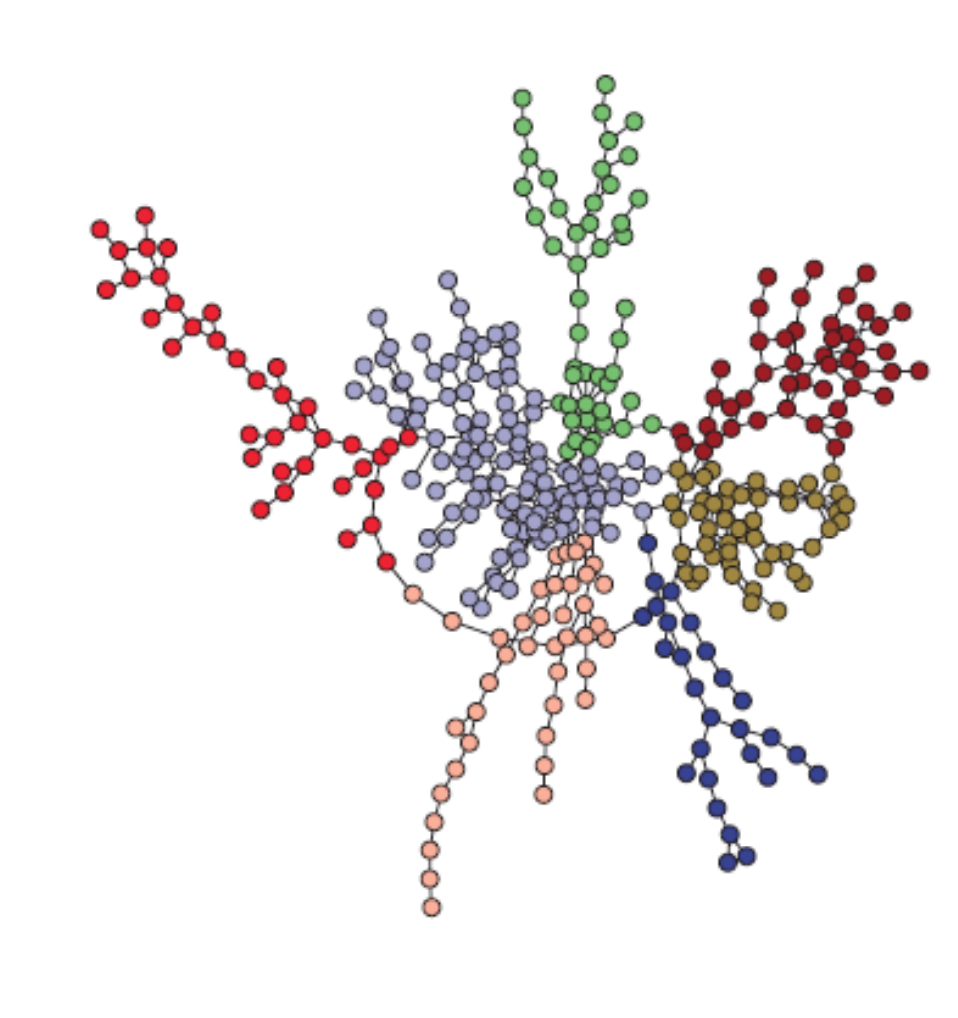
**VISUALIZATION OF RESULTS:**



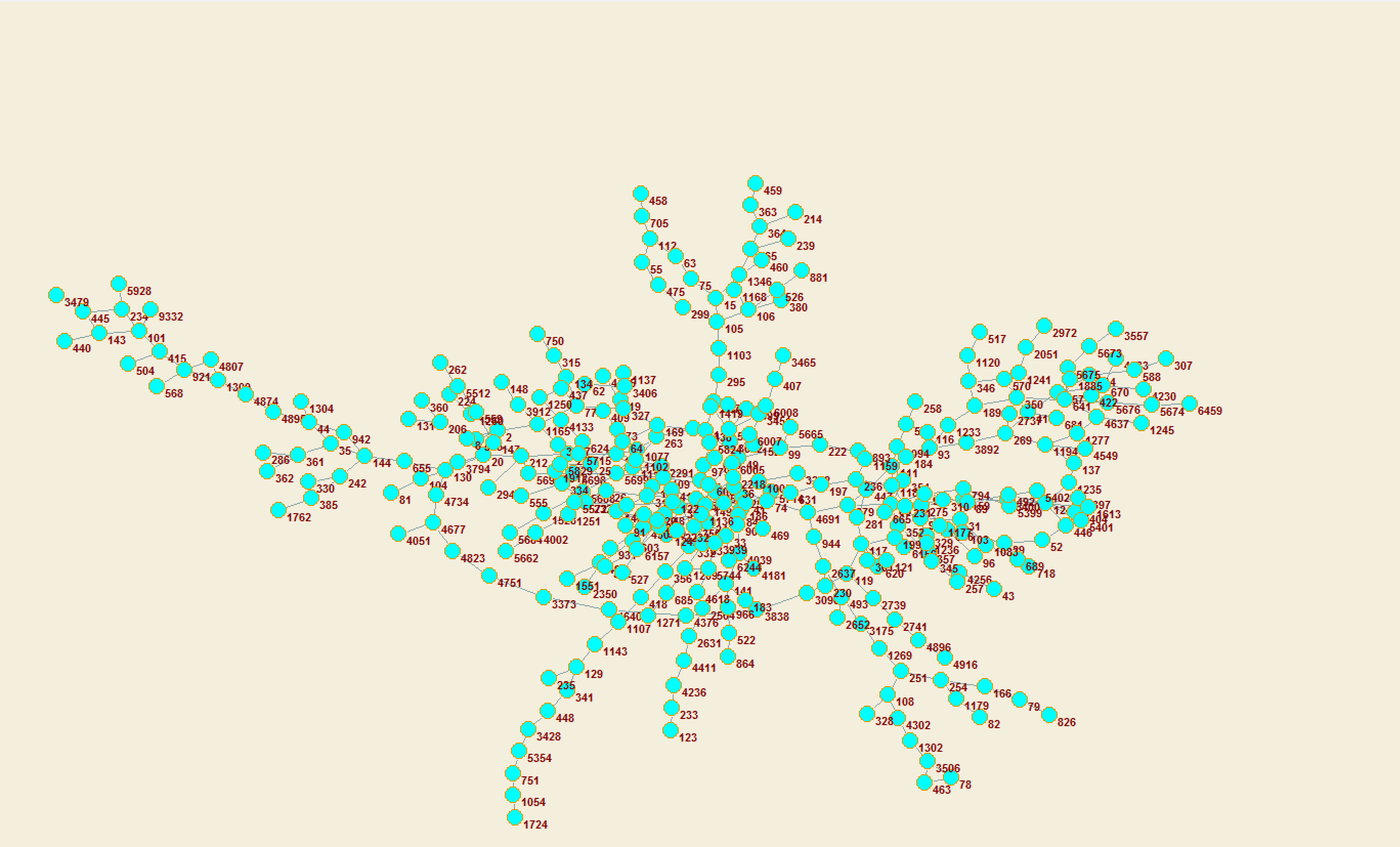
**RESULTS AND VISUALIZATION FOR DATASET 3:**

* Setting the expansion and inflation parameters to 28 yielded the best steady state results and the results thus obtained are displayed below as follows.
* Any Parameter setting after 28 seemed to have given more or less the same visualization.

**VISUALIZATION BASED ON GROUNDTRUTH:**



VISUALIZATION OF RESULTS:



DRAWBACKS AND ADVANTAGES:

**ADVANTAGES:**

* Scales well with increasing graph size.
* Works with both weighted and non-weighted graphs.
* Produces good clustering results.
* Robust against noise in graph data.
* Number of clusters not specified ahead of time, but can adjust cluster granularity with parameters.

**DISADVANTAGES:**

* Cannot find overlapping clusters (in general).
* Not suitable for clusters with large diameter.

RUNTIME:

* The runtime for the algorithm is O(Nexpansion or Ninflation) where N is the size of the input matrix and expansion and inflation are the parameters set and whichever is greater.
* There are different sections of the code that run independent of each other in O(N2) time.
* For Generating Input Matrix: O(n)
* For Indexing File: O(n)
* For creating mapping of final clusters: O(n2)

TOATAL RUNTIME: O(n) + O(n) + O(n2) + O(N) = O(Nexpansion or Ninflation).

README FOR EXECUTION OF CODE:

* There are three code files each labelled with the dataset they use
* Please change the path for the output file in each of the files to get the corresponding output file
* The files can be run from the terminal and use Python 2.7.11
* Please adjust the expansion and inflation parameters in the Terminal and run the files as necessary.