Social network analysis – final project

Part 1 – Graps details:

1. **Game of thrones:**
   - Link: https://github.com/melaniewalsh/sample-social-network-datasets/tree/master/sample-datasets/game-of-thrones
   - Nodes: 107
   - Edges: 352
   - Graph density: 0.062
   - Network description: The nodes csv contains 107 different characters, and the edges csv contains 353 relationships between those characters, which were calculated based on how many times two characters' names appeared within 15 words of one another in the novel.

2. **email-Eu-core**
   - link : https://snap.stanford.edu/data/email-Eu-core.html
   - Nodes: 1,005
   - Edges: 25,571
   - Graph density: 0.002
   - Network description:The network was generated using email data from a large European research institution.There are anonymized information about all incoming and outgoing email between members of the research institution. There is an edge (u, v) in the network if person u sent person v at least one email.

3. **LastFM Asia Social Network**
   - Link: https://snap.stanford.edu/data/feather-lastfm-social.html
   - Nodes: 7,624
   - Edges: 27,806
   - Graph density: 0
   - Network description: Nodes are LastFM users from Asian countries and edges are mutual follower relationships between them. The vertex features are extracted based on the artists liked by the users. The task related to the graph is multinomial node classification - one has to predict the location of users. This target feature was derived from the country field for each user.

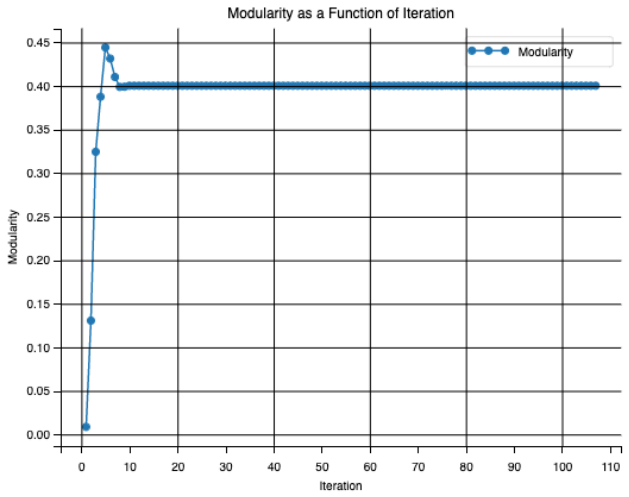
Part 2 – Graph Modularity – regular version Vs using Freezing mechanism:

All graphs calculations used a seed value of 5.

1. Game Of Thrones:

Graph of modularity as a function of iterations:

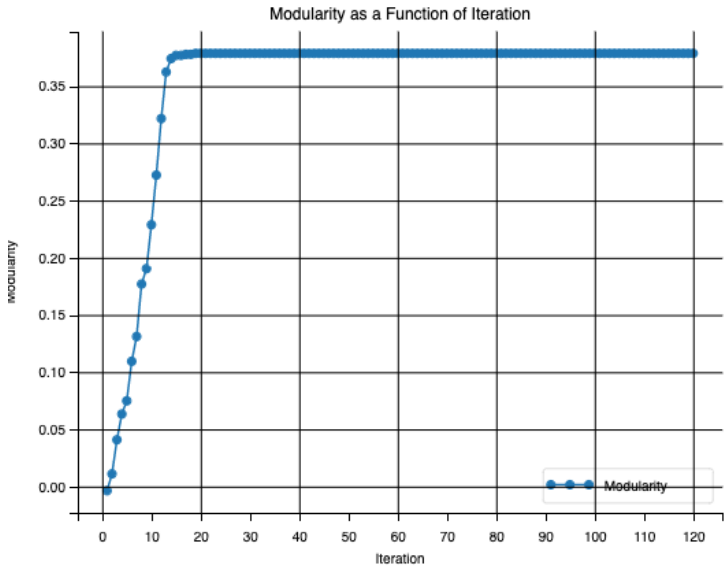
Regular algorithm version:



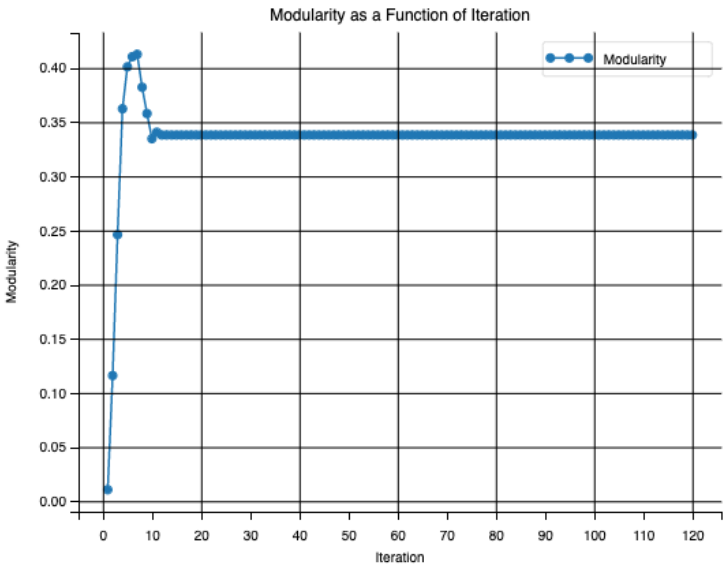
The modularity starts off low but quickly rises, showing that the algorithm is improving the community structure. Although there are some ups and downs along the way, it generally keeps going up until it converges to a stable value after a few iterations(approximately 10).

Algorithm with freezing mechanism:

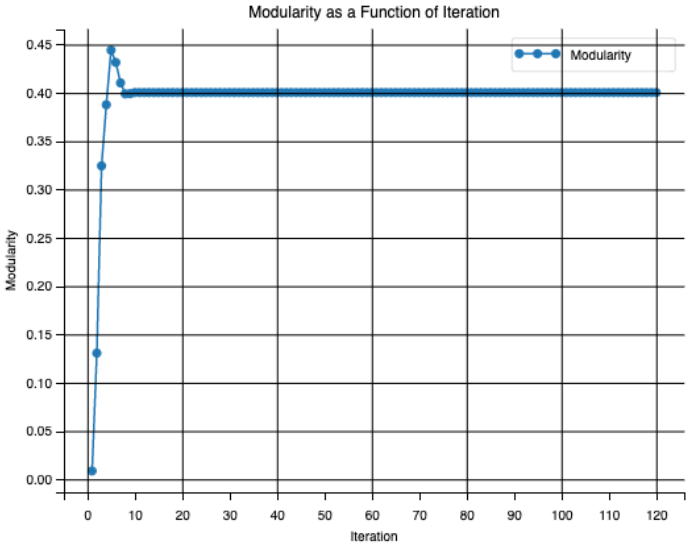
Freezing = 0.1:



Freezing = 0.5 :



Freezing = 0.9:

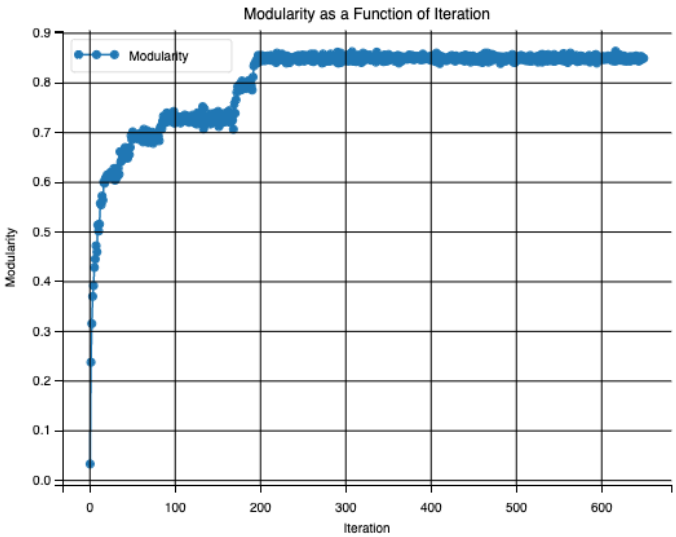


using the freezing mechanism with a low percentage (0.1), we see a moderate increase in modularity initially. Unlike the original graph, we don't encounter a decline. rather, we observe a slight increase followed by convergence to a value lower than the regular algorithm modularity value.
Increasing the freezing percentage to 0.5 shows a similar pattern to the original algorithm, with modularity values close to what we achieved without freezing. This could be useful for faster running times.
With a high freezing percentage (0.9), the graph behaves almost the same as the original algorithm without freezing.

2. European email:

Graph of modularity asa function of iterations:

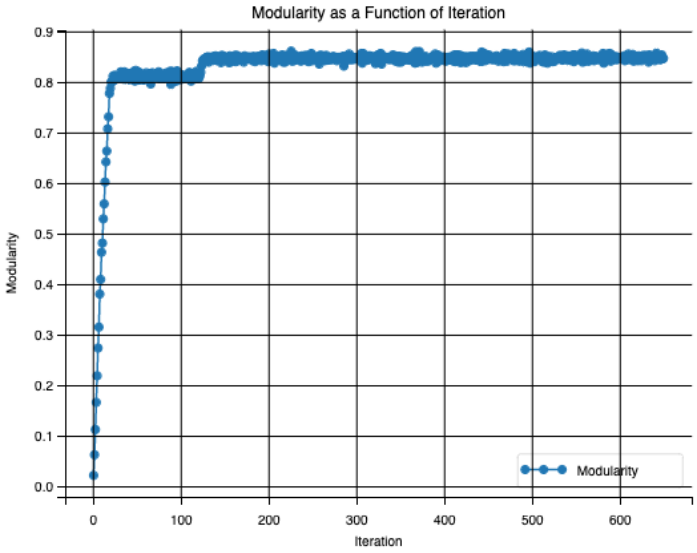
Regular algorithm version:



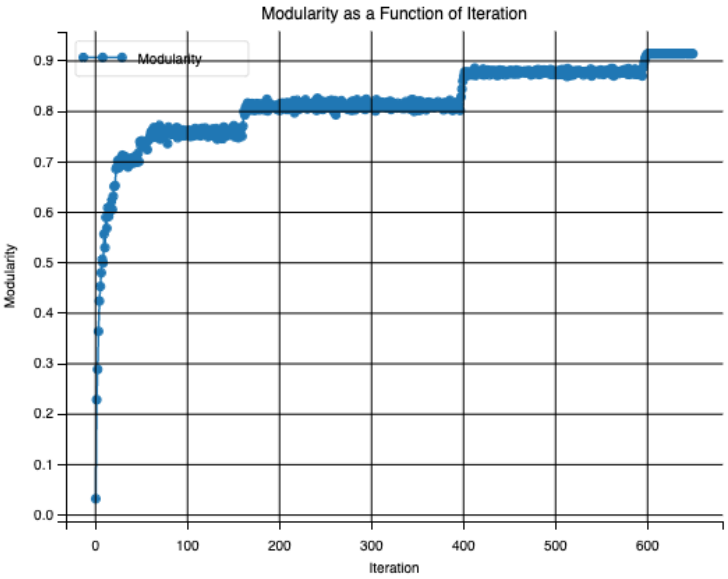
In this scenario, we observe two distinct patterns of rapid growth. Initially, there's a sharp increase from iteration 0 to iteration 100. Subsequently, it appears to converge, but around iteration 180, there's another notable surge. However, by the 200th iteration, we observe convergence towards a specific value.

algorithm with freezing mechanism version:

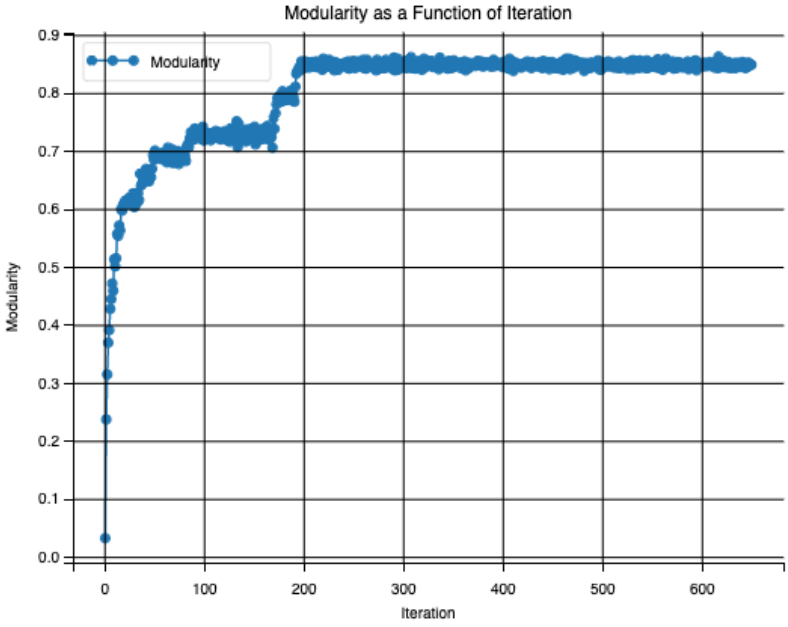
freezing = 0.1



Freezing = 0.5



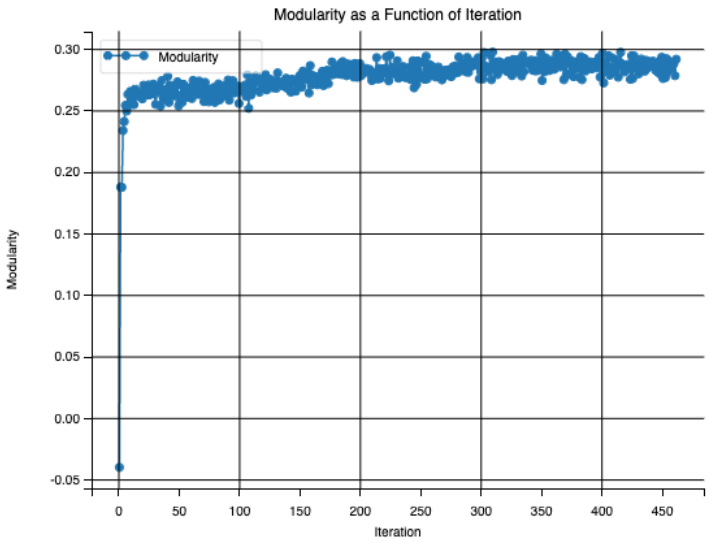
freezing = 0.9



In the case of applying freezing, even a low percentage yields results quite similar to those of the original algorithm.
Furthermore, it's evident that as the freezing percentage increases, the graph's behavior becomes more stabilized, eventually resembling closely the graph generated by the original algorithm. Eventually, it produces outcomes analogous to those of the original algorithm.In this scenario, the algorithm terminates not due to reaching a specific iteration count, but rather upon reaching a certain threshold of label changes for the nodes.

3. LastFM

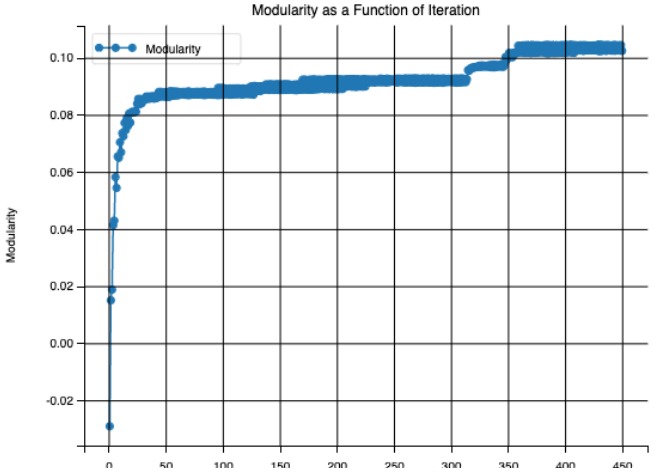
Regular algorithm version:



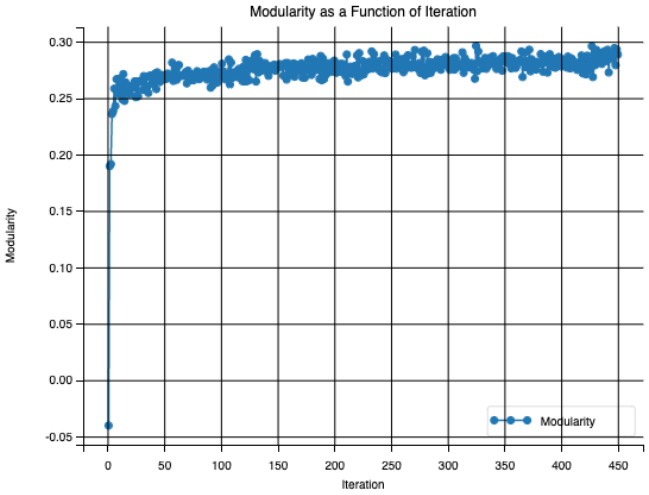
In this scenario, we observe an extremely rapid surge in modularity within a few initial iterations. However, following this initial spike, there appears to be a convergence, albeit not entirely smooth. Small fluctuations in modularity values are still noticeable, indicating a degree of instability rather than a complete convergence.

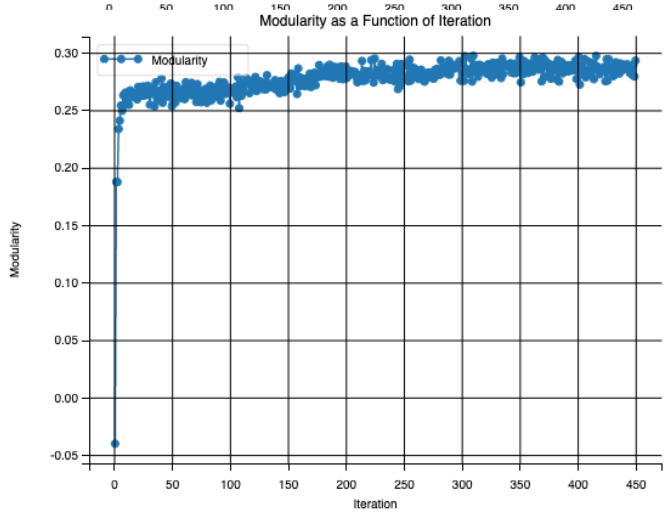
Algoritm with freezing mechanism:

Freezing = 0.1



Freezing = 0.5





using the freezing mechanism, a lower freezing percentage yields a more stable graph, albeit failing to converge to the original algorithm's modularity value. However, with freezing percentages of 0.5 and 0.9, we observe similar outcomes to the original algorithm. Notably, the latter exhibits more moderate modularity value fluctuations compared to the former.

Freezing = 0.9

Part 3 – grpah best modularity:

1. **Game of thrones:**
   A good community partition I found:
   - Number of communities: 6
   - Modularity score: 0.4003

2. **European email:**
   A good community partition I found:
   - Number of communities: 37
   - Modularity score: 0.8570

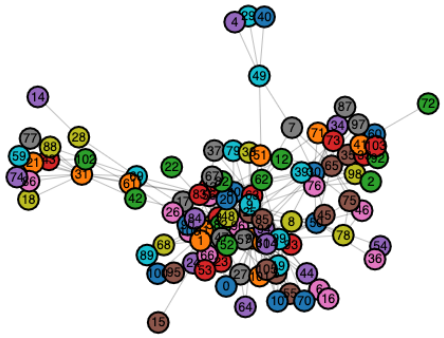
3. **Last FM:**
   A good community partition I found:
   - Number of communities: 27
   - Modularity score: 0.2942

Part 4 – algorithm results on the graphs:

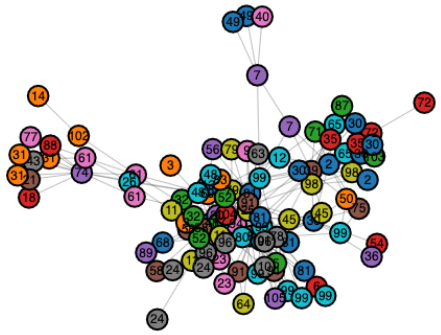
1. **Game of thrones:**

As it is a “School example” graph, I will also illustrate the algorithm's progression, highlighting the evolution of node labels across iterations.

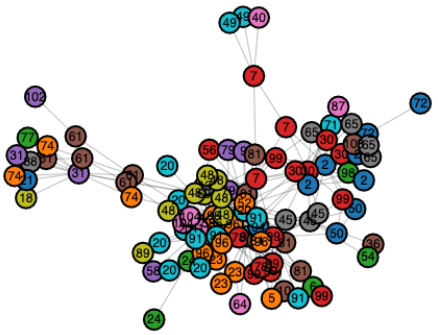
Initial state



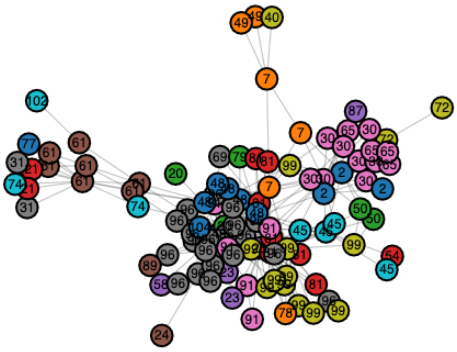
Iteration: 1



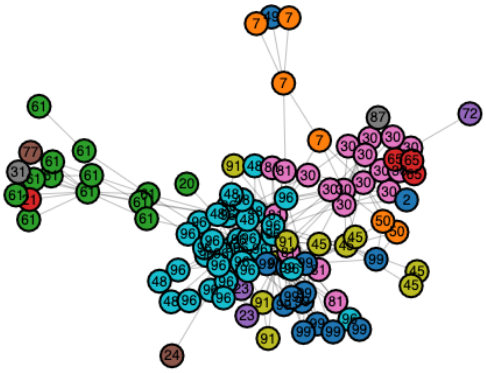
Iteration: 2



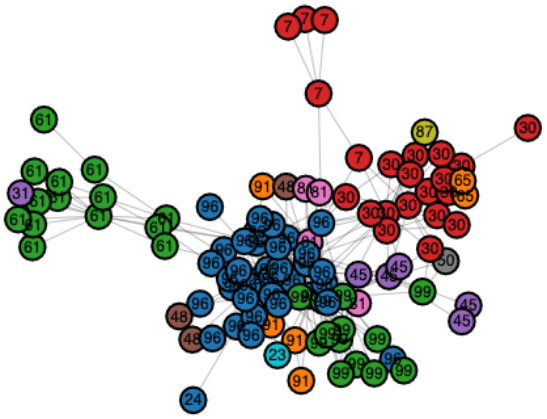
Iteration: 3



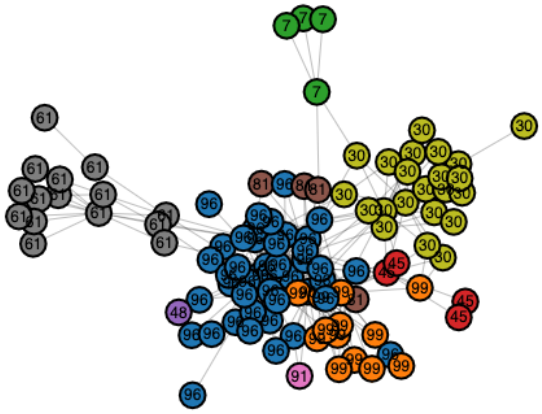
Iteration: 4



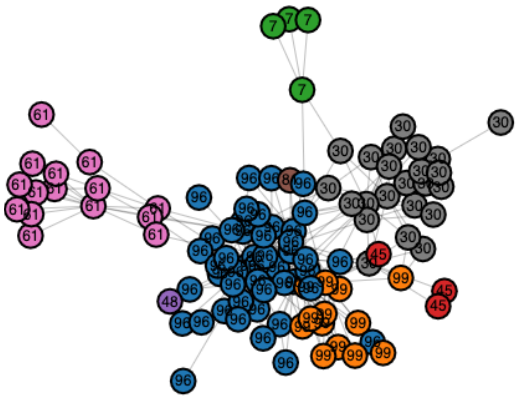
Iteration: 5



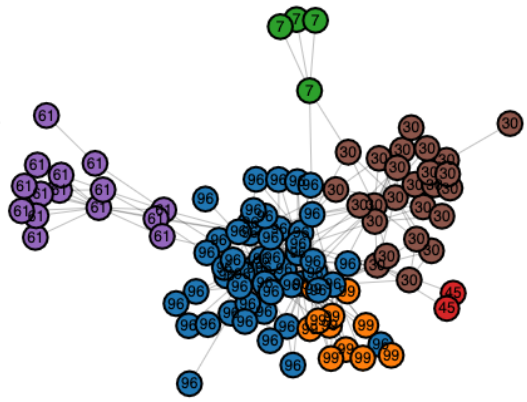
Iteration: 6

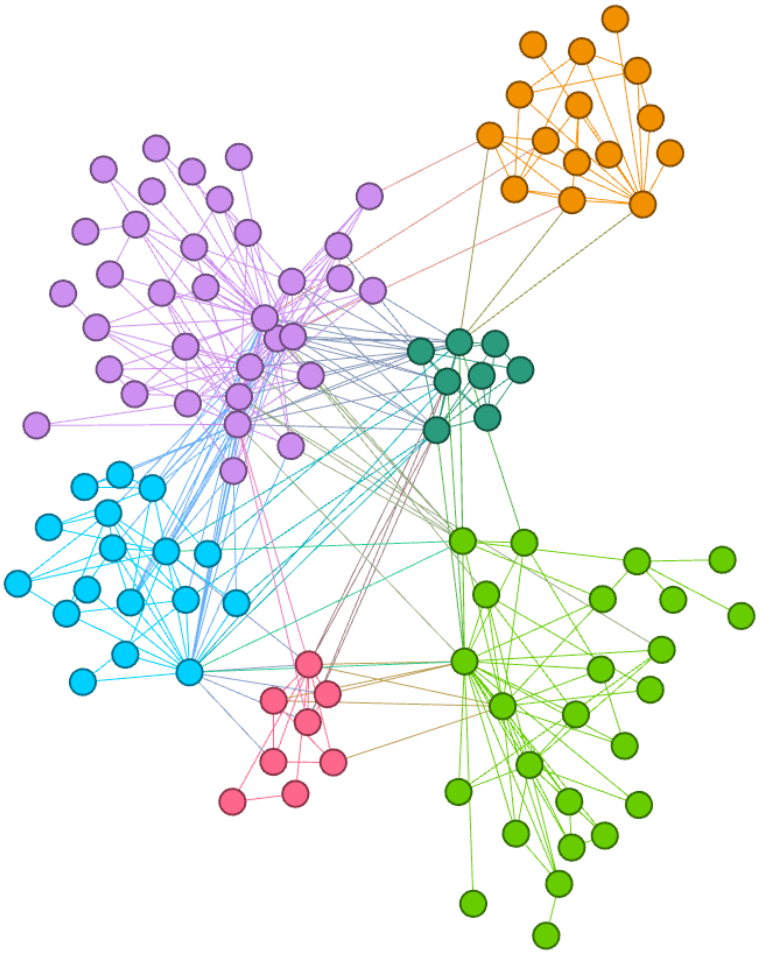


Iteration: 7



Iteration: 8



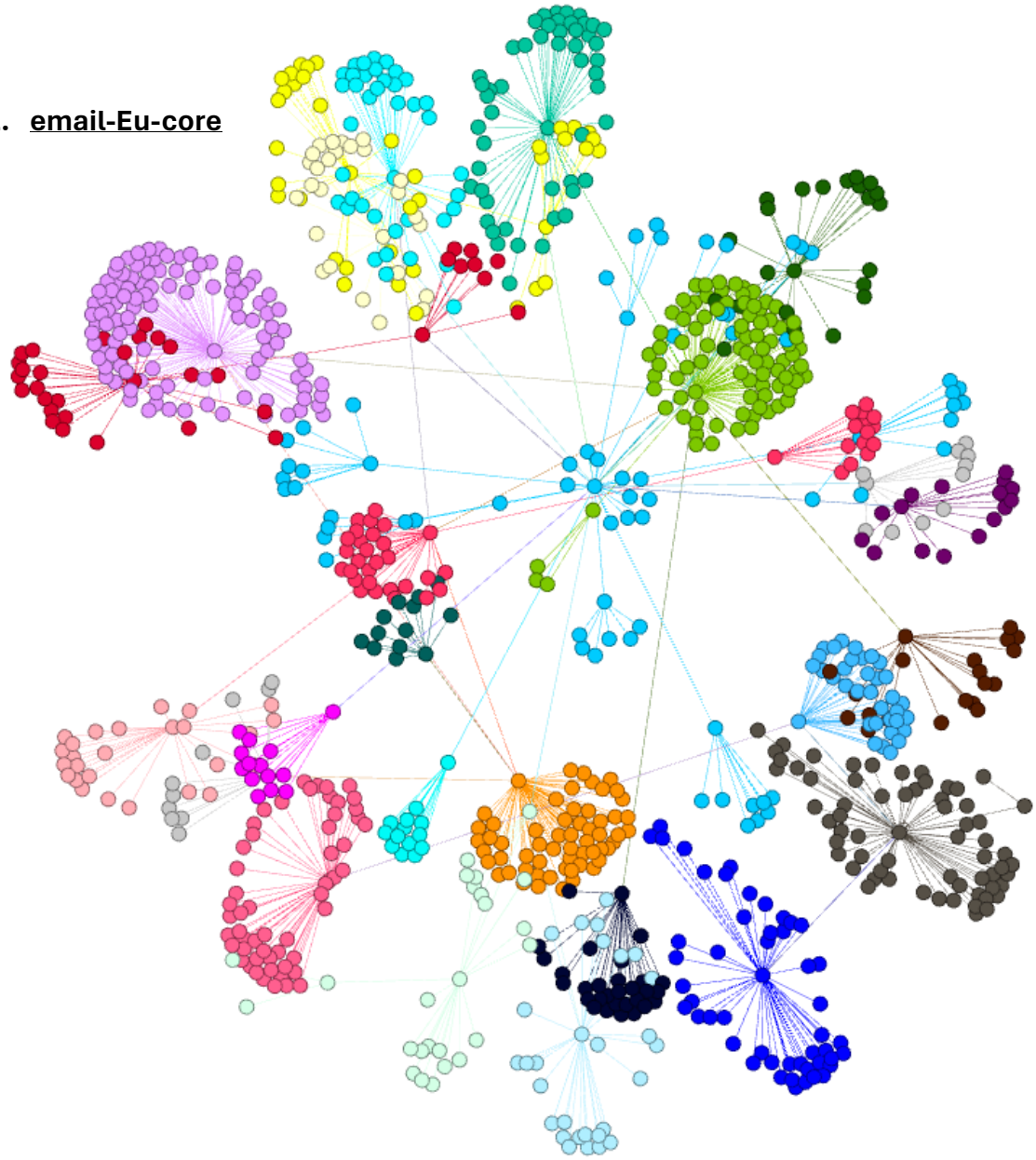


The graph shows six different Comuunities. We can notice fewer edges linking these communities compared to the edges within each community. Additionally, within each community, there is a central node that connects most of the other nodes.

Closeness Centrality: top 10 nodes

Node Id	Closeness Centrality
3	0.51207729
9	0.50961538
56	0.5
99	0.48847926
91	0.48623853
58	0.47963801
30	0.47963801
62	0.47963801
96	0.46902655
85	0.46086957

2. email-Eu-core

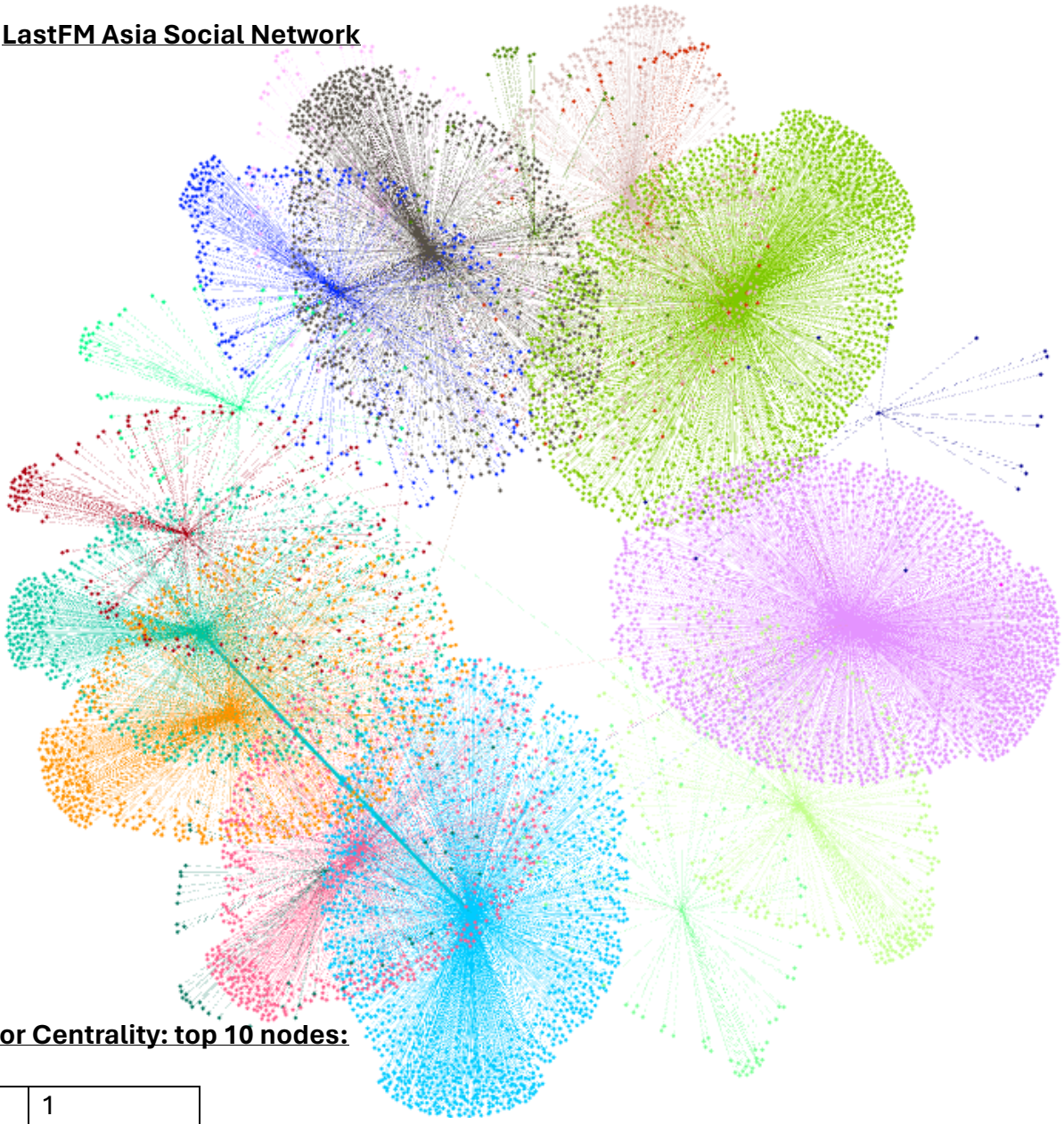


Betweenness Centrality: top 10 nodes

Node id	Betweenness Centrality
14	0.04152779
11	0.01235406
4	0.05026254
9	0.01461835
21	0.02801782
25	0.00305809
7	0.02400893
10	0.01850958
34	0.00635229
26	0.00452763

In this graph, there are 37 communities with 1005 nodes. The number of communities is relatively high. Within each community, there are a few key nodes to which most other vertices are connected. Moreover, the number of edges between communities is small compared to the total number of edges in the graph.

3. LastFM Asia Social Network



Eigenvector Centrality: top 10 nodes:

17	1
10	0.71180103
0	0.55209567
6	0.2498049
14	0.22523584
3	0.20925242
8	0.16925277
5	0.14717803
15	0.09813807
16	0.09094226

In this graph, there are 27 communities with 7624 nodes. Although it has the highest number of vertices among the three graphs, it has fewer communities compared to the second graph. Each community in this graph exhibits a distinct central hub containing multiple vertices, from which numerous edges emanate within the community, resulting in cohesive clusters. Additionally, the number of edges crossing between communities is notably lower than the total number of edges in the entire graph.

Conclusions :

The work provides insights into community detection algorithms, particularly focusing on label propagation.
It explores the impact of parameters like freezing thresholds and maximum iterations on modularity and community structure.
Through experimentation on various network datasets, the assignment reveals patterns of modularity convergence and community formation dynamics. The findings suggest that adjusting algorithm parameters can significantly influence community detection outcomes, highlighting the importance of parameter tuning for effective network analysis.

Furthermore, the course of my work has yielded key insights. Initially, the implemented algorithm consistently delivers results, yet I encountered an issue regarding the accuracy of modularity calculations during execution on large scale graphs.

However, integrating the algorithm with the freezing mechanism provides notable runtime optimizations while maintaining nearly identical results to the original algorithm. Across multiple runs with the freezing mechanism, the final modularity values reliably converged to levels comparable to those obtained without freezing. This highlights the potential of leveraging the freezing mechanism for runtime efficiency without compromising result fidelity.

Also, I found that in algorithms reliant on probability, maintaining a fixed seed is crucial to ensure reproducibility. I observed that without a fixed seed, results varied significantly, sometimes with no apparent correlation between them. Additionally, I noted that the algorithm demonstrates relative success in convergence, efficiently identifying communities compared to the overall size of the network.

Demonstration of code operation

1. **Run the main.py**
2. **Choose the desire graph.**
3. **Choose if you want to determine a seed value.**
4. **Choose if you want to add the freezing mechanism.**
5. **Choose if you want to use default thresholds.**
    - 5.1. **If yes, choose which threshold you want to determine.**
6. **The main menu will be show:**
    - 6.1. **1 - run the algorithm.**
        2 – set seed
        3 – print modularity per iteration
        4 - change freezing threshold
        5 - cancel freezing threshold
        6 - change max-iteration threshold
        7 - change label's changes threshold
        8 - change graph
        9 - print communities
        10 - print graph view
7. **At any time press 0 to exit.**