Q7. Social Media Network Analysis

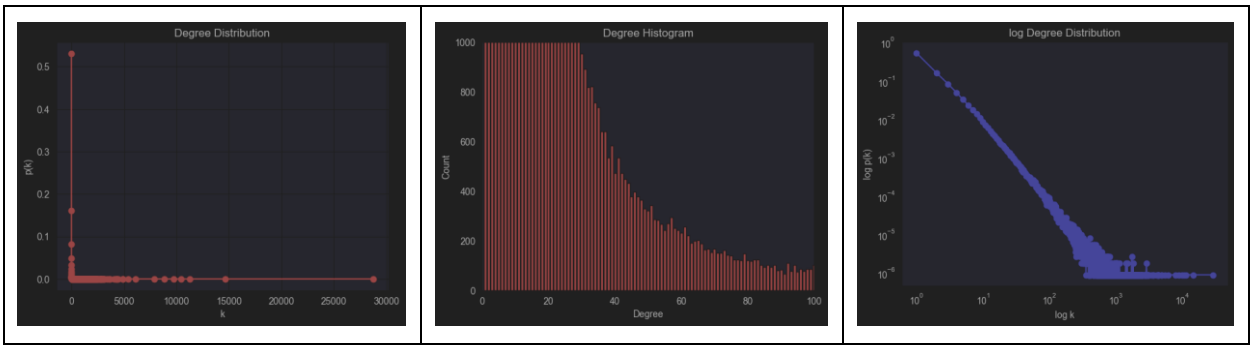
Reference

https://www.kaggle.com/code/lodetomasi1995/complete-graph-analysis-sbm-k-core-dec-er

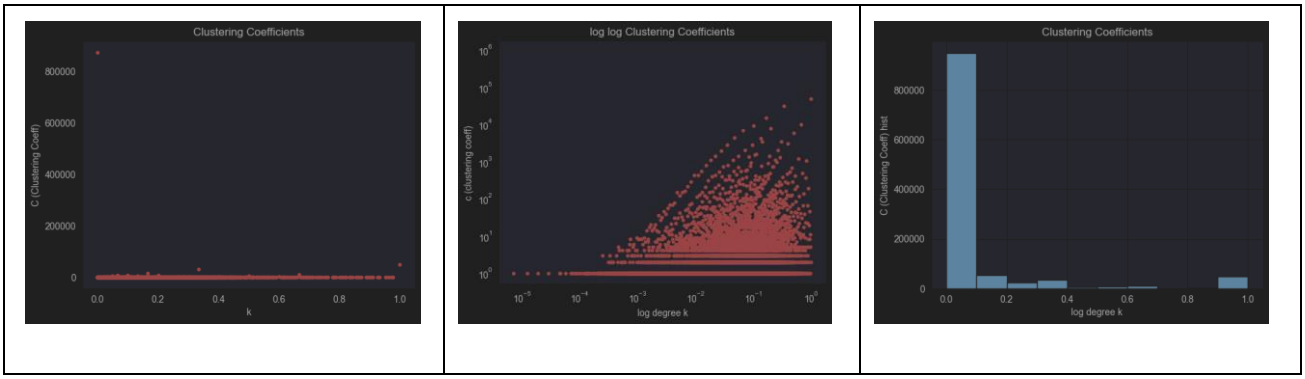
https://arxiv.org/abs/1205.6233

a. Calculate and plot the clustering coefficient and degree distribution

We recall that the degree of a node is the number of directly connected neighbors of the node. For any integer k>=0, the quantity p_k is the fraction of nodes having degree k. This is also the probability that a randomly chosen node in the network has degree k. The quantities pk, for k =>0, represent the degree distribution of the network.



The Clustering Clustering (or Transitivity) is used as a way measure the degree of connectedness, as well as a way to study the large-scale structure of a network. The clustering coefficient C_iof a vertex i is can be interpreted as the propensity a node has to form triangles

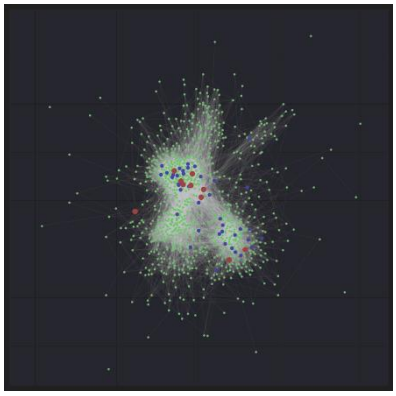


b. Identify the most influential nodes

If there are many other nodes connected to a node, then the latter can be considered important. Therefore, the centrality of a node can be measured based on its degree: $DC = \frac{N_{degree}}{n-1}$

Node centrality refers to the number of connections between a node and other nodes, which is the degree of that node. In a network graph, nodes with higher degrees are usually considered important nodes because they have connections with many other nodes and can quickly transmit and receive information.

Top 5 nodes with highest degree centrality:
Node: 1072 Centrality: 0.025536398537654344
Node: 363 Centrality: 0.012900821137573806
Node: 35661 Centrality: 0.009940179171707541
Node: 106 Centrality: 0.009217641549085418
Node: 482709 Centrality: 0.008601722282972167



The graph consists of one thousand points, with the 10 points with the highest degree centrality marked in red, followed by 50 points in blue, and the rest in green. It is not difficult to find that influential points often gather together because they play a role similar to a "hub" or "bridge" in the network. They can connect different communities or groups, allowing information to flow quickly through the network. Meanwhile, these nodes may also have some shared features or attributes that make them more inclined to connect with each other.

For example, in social networks, nodes with high degree centrality may be influential individuals who typically have many followers and followers, and their opinions and behaviors can quickly spread across the network. These influential figures may also have some common characteristics, such as being experts in a certain field or core members of a community, so they are more inclined to pay attention to each other and form an "elite circle".

c. Identify Isolated Nodes in the Network

We find that there is no isolated nodes.

```
# Find isolated nodes
isolated_nodes = list(nx.isolates(G))

# Print isolated nodes
print("Isolated nodes: ", isolated_nodes)
```
Executed at 2023.12.16 01:06:16 in 195ms

```
Isolated nodes:  []
```

d. Recognize Connected Components in the Network

Since no isolated nodes:

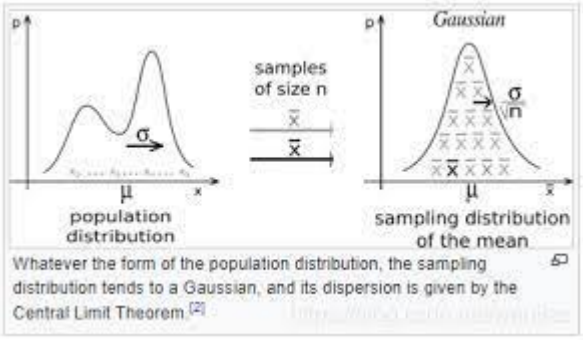
```
# Find connected components
connected_components = [len(c_components) for c_components in sorted(nx.connected_components(G),
 key = len, reverse  = True)]

# Print count of connected components
print("connected components: %s" % connected_components)
```
Executed at 2023.12.16 01:15:37 in 1s 123ms

```
connected components: [1134890]
```

e. Compute Average Shortest Path Length of the Network.

The geodesic path between two nodes in a graphs with minimum number of edges is called shortest path. If the graphs is weighted, the path is the minimum sum of edge weights. The geodesic distance is not necessarily unique, but geodesic is well defined since all geodesic paths have the same length. The idea behind short distance is that number of nodes one encounters in a breath-first visit of a graph starting from source increase exponentially with the distance to source node, hence the distance increase logarithmically in the number of visited nodes.

Due to the large size of the graph, we can only adopt some approximation methods. Here, we randomly sample 4000 pairs of nodes each time, calculate 50 times, and finally take the average. The basis for doing this is the law of large numbers and the central limit theorem. If we sample enough and large, we can approach the true value.



Whatever the form of the population distribution, the sampling distribution tends to a Gaussian, and its dispersion is given by the Central Limit Theorem [2]

Results got after running for 25 minutes:

Variance: 0.0007781404000000021

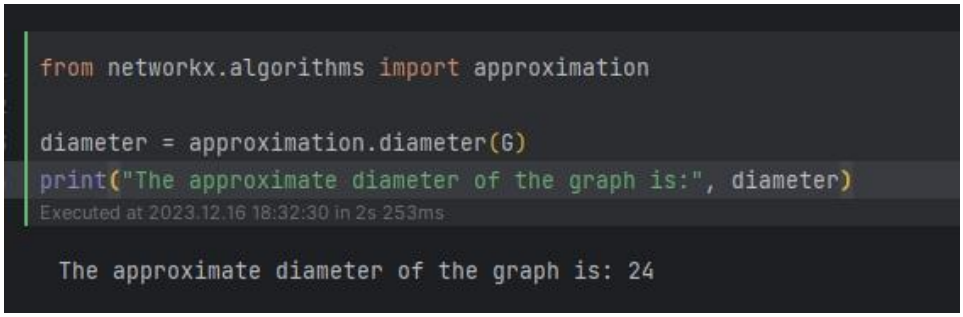
The average shortest path length of the network is: 5.28136

f. Calculate the Diameter of the Network

In graph theory, the diameter of a graph is defined as the maximum length of the shortest path between all pairs of vertices in the graph. In other words, it measures the farthest possible distance between any two nodes in the graph.

However, for very large graphs, calculating the actual diameter can be very time-consuming, so approximate methods are often used to estimate this value. We use the *approximation.diameter ()* method to calculate the approximate diameter of Figure G.

The *approximation.diameter* function of *NetworkX* uses an approximation algorithm called the *Whitepath* algorithm to estimate the diameter of the graph.



g. Detect Community Structures

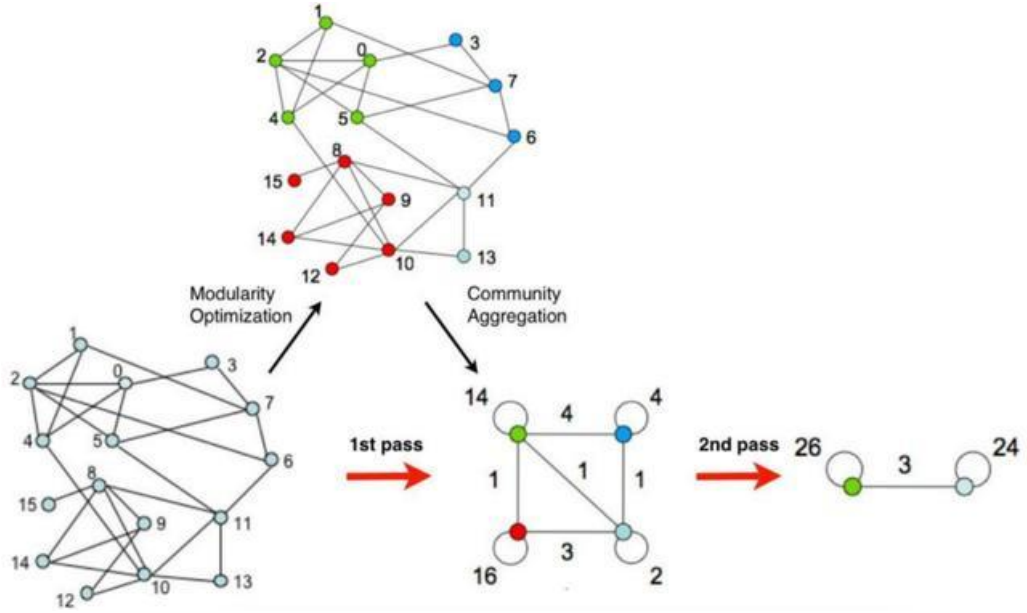
Louvain algorithm

A modularity based graph algorithm model, which differs from ordinary modularity based and modularity based gains in that the algorithm is fast and has particularly noticeable clustering effects on graphs with few polygons.

Algorithm process:

1. Initially, treat each vertex as a community, with the same number of communities as the number of vertices.

2. Merge each vertex with its adjacent vertices in sequence, calculate whether their modularity gain is greater than 0, and if it is greater than 0, place the node in the community where the adjacent node is located.
3. Iterate the second step until the algorithm stabilizes, meaning that the community to which all vertices belong no longer changes.
4. Compress all nodes in each community into one node, convert the weight of points within the community into the weight of the new node ring, and convert the weight between communities into the weight of the new node edge.
5. Repeat steps 1-3 until the algorithm stabilizes.



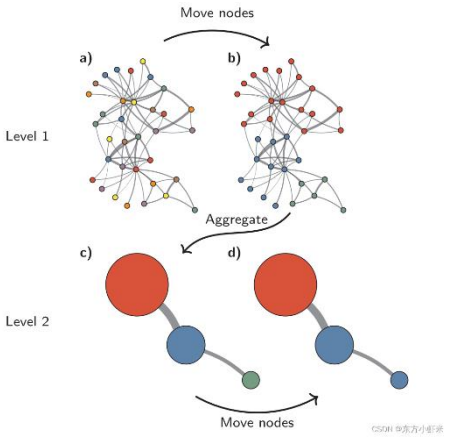
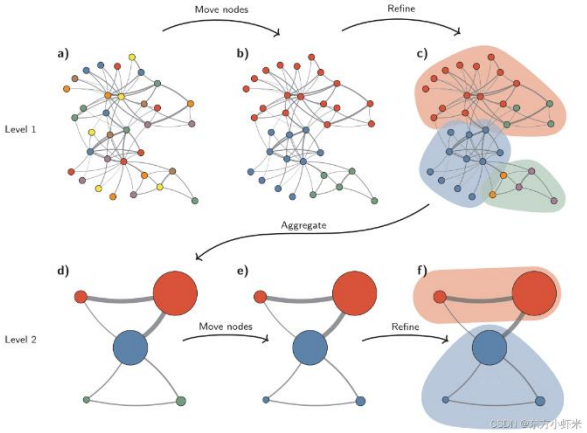
Leiden algorithm

The Leiden algorithm is one of the state-of-the-art algorithms in recent years.

The Louvain algorithm has a major flaw: it may generate any community with poor connectivity (even disconnected). To solve this problem, the author introduced the Leiden algorithm. It has been proven that the community guarantee generated by this algorithm is connected. Furthermore, it has been proven that when the Leiden algorithm is iteratively applied, it converges to a partition where all subsets of all communities are locally optimal allocations. And the algorithm speed is faster than the Louvain algorithm.

Louvain provides two guarantees: (1) no communities can merge, and (2) no nodes can move. (The algorithm is iterated multiple times, with the previous algorithm's community partitioning result serving as the starting point for the next iteration until further improvement is not possible.)

However, multiple iterations of the Louvain algorithm can make the problem of poor connections more apparent, as the next iteration may move the bridge node (as shown in node 0 in the previous round) to another community, resulting in poorer connections within the community.

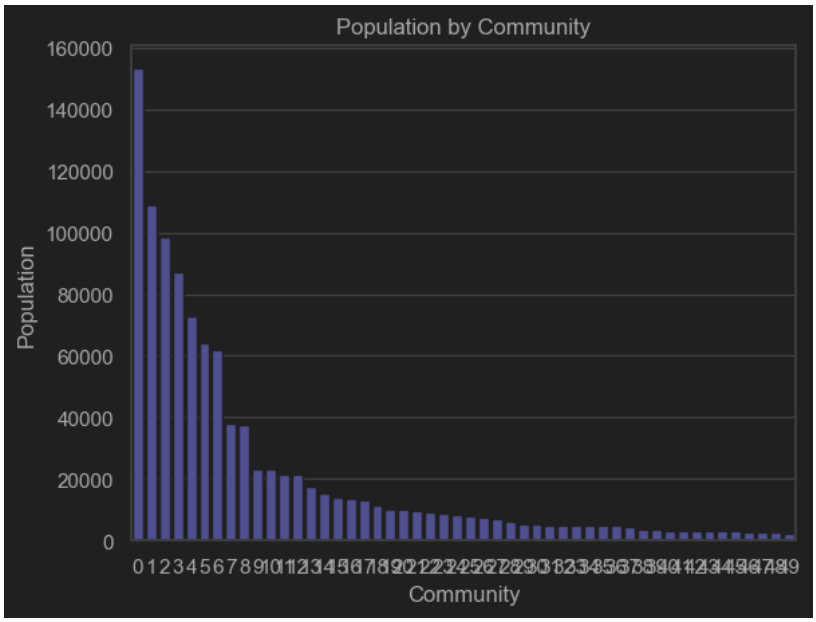
Louvain algorithm	Leiden algorithm
	

Analysis and visualization

Using the *leidenalg* library for clustering, the number of communities obtained under default parameters is:

The number of communities: 4829

The number of people in different communities is shown in the figure:



All the information is saved in the text file: communities.txt

When using the Leiden algorithm for community detection, the parameters that can be adjusted are as follows:

1. *Partition_Type*: Partition type. You can choose different partition types, such as leidenalg ModularyVertexPartition, leidenalg RBConfiguration VertexPartition, etc.
2. *Weights*: Edge weights. Weights can be assigned to edges to take into account their weight in community detection. By default, the weight of all edges is 1.
3. *Resolution_Parameter*: Resolution parameter. This is a floating-point number used to adjust the resolution of community detection. Higher resolution parameters lead to smaller communities, while lower resolution parameters lead to larger communities. The default value is 1.0.

Here is a graph of two thousand nodes, with different colors representing different communities.