EE447 Mobile Internet Lab 3

Zhou Litao 518030910407 F1803016

May 22, 2021, Spring Semester

1 Introduction

Modern networks are growing exponentially in size, diversity and complexity. Due to the changes in networks, a wide variety of networks are emerging, such as IoT data, wireless sensor data, cloud data, co-citation in academic fields, and social network data. A community in a network is composed of a set of nodes that are highly connected to each other, unlike other nodes in the network that have relatively random and scattered relationships. A key role of community detection algorithms is that they can be used to extract useful information from the network.

The modularity is used to measure whether the division of a community is a relatively good result. A relatively good result has a high similarity of nodes inside the community and a low similarity of nodes outside the community.

$$Q = \frac{1}{2\ \mathrm{m}} \sum_{i \neq j} \left(A_{ij} - \frac{k_i k_j}{2m} \right) \delta\left(C_i, C_j\right)$$

A_{ij} is an element of the adjacency matrix of the network, C_i, C_j denotes the two communities where node i and node j are located respectively, and m is the total number of edges in the network, k_i denotes the degree of node i. If node i and node j are in the same community, $\delta\left(C_i, C_j\right)$ is 1, otherwise $\delta\left(C_i, C_j\right)$ is 0 .

2 Purpose

This Lab focuses on the community detection algorithms for networks. In this Lab, we will use python to complete community detection under a specific network and visualize the network based on the results.

To ensure the quality of visualization, we choose to generate a random graph for the graph clustering algorithm. As for the choice of the community detection algorithm, we will implement the Louvain's algorithm, which aims to maximize the modularity metric mentioned above.

3 Louvain's Algorithm

Louvain Algorithm is a greedy and local search algorithm maximizing the modularity. The algorithm runs in terms of passes. Each pass is made of 2 phases:

- Phase 1: Modularity is optimized by allowing only local changes of communities
- Phase 2: The identified communities are aggregated in order to build a new network of communities
- Go to Phase 1

The passes are repeated iteratively until no increase of modularity is gained.

3.1 Phase 1 Partitioning

1. For each node i, the algorithm performs two calculations:
    - Compute the modularity gain (ΔQ) when putting node i from its current community into the community of some neighbour j of i

- Move i to a community that yields the largest modularity gain (ΔQ)

2. The loop runs until no movement yields a gain, to be specific, the gain is calculated as follows, which is basically derived from the modularity definition. If we move node i to community C, ΔQ can be calculated using local information as follows

$$\Delta \mathrm{Q}(\mathrm{i} \rightarrow \mathrm{C}) = \left[\frac{\sum_{in} + k_{i,in}}{2m} - \left(\frac{\sum_{tot} + k_i}{2m}\right)^2\right] - \left[\frac{\sum_{in}}{2m} - \left(\frac{\sum_{tot}}{2m}\right)^2 - \left(\frac{k_i}{2m}\right)^2\right]$$

- Σ_{in} the sum of the weights of the links inside C
- Σ_{tot} the sum of the weights of all links to nodes in C
- k_i the sum of the weights (i.e., degree) of all links to node i
- $k_{i,in}$, the sum of the weights of links from node i to nodes in C
- m is the sum of the weights of all edges in the graph

3. We also need to compute ΔQ(D → i) of taking node i out of community D, and the gain of a movement is a sum of the two terms.

$$\Delta \mathrm{Q} = \Delta \mathrm{Q}(\mathrm{i} \rightarrow \mathrm{C}) + \Delta \mathrm{Q}(\mathrm{D} \rightarrow \mathrm{i})$$

3.2 Phase 2 Restructuring

The partitions obtained in the 1st phase are contracted into super- nodes, and the weighted network is created as follows:

1. Super-nodes are connected if there is at least one edge between nodes of the corresponding communities
2. The weight of the edge between the two super-nodes is the sum of the weights from all edges between their corresponding partitions The loop runs until the community configuration does not change anymore

An interpretation of the algorithm can be found in Figure 1.

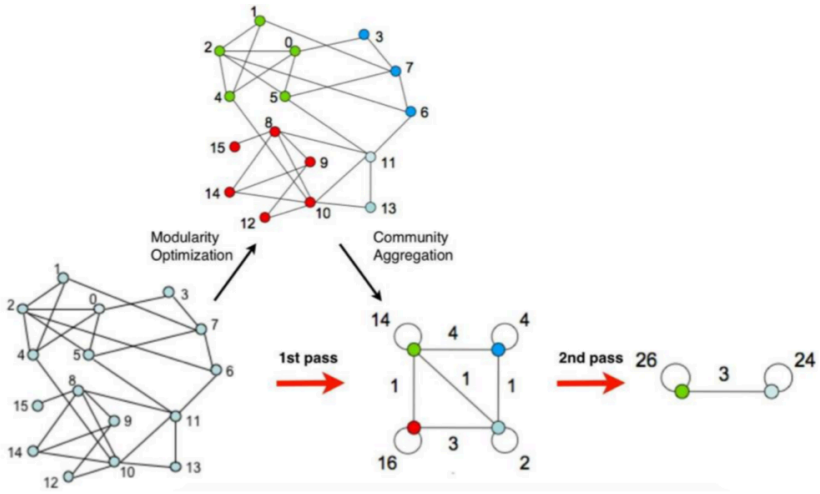

Figure 1: Interpretation of Louvain's Algorithm

4 Experiment

We implement Louvain's Algorithm in Python, which requires the following packages.

- `networkx` a package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks.
- `matplotlib` for visualization of the community results

As for the graph data, we generate them from `networkx.powerlaw_cluster_graph`, which uses Holme and Kim algorithm for growing graphs with powerlaw degree distribution and approximate average clustering. We generate a graph with 100 nodes, with average degree of 2 and probability of adding a triangle after adding a random edge to be 0.8. We perform the Louvain's algorithm on this random graph. The running output and visualization of the community result can be found in Figure 2 and 3.

```
~/GIT/2021Spring main* 11s
Base ) /opt/anaconda3/bin/python /Users/ltzhou/GIT/2021Spring/EE447/community/main.py
Round 0, (196, 100)
New Phase, modularity -0.023935339441899222
Iteration #1, 69 vertices Moved, modularity: 0.30275145772594736, cluster#: 44
Iteration #2, 24 vertices Moved, modularity: 0.3916467097420966, cluster#: 32
Iteration #3, 11 vertices Moved, modularity: 0.4214910453977508, cluster#: 29
Iteration #4, 8 vertices Moved, modularity: 0.4217253227821738 4, cluster#: 27
Iteration #5, 7 vertices Moved, modularity: 0.4383590170762181, cluster#: 26
Iteration #6, 4 vertices Moved, modularity: 0.4319424198250727, cluster#: 26
26 clusters
Round 1, (196, 26)
New Phase, modularity 0.4319424198250727
Iteration #1, 20 vertices Moved, modularity: 0.5235188463140358, cluster#: 10
Iteration #2, 9 vertices Moved, modularity: 0.5706346314035818, cluster#: 8
Iteration #3, 3 vertices Moved, modularity: 0.5715087255310288, cluster#: 7
Iteration #4, 2 vertices Moved, modularity: 0.585407122032 4867, cluster#: 6
Iteration #5, 0 vertices Moved, modularity: 0.585407122032 4867, cluster#: 6
6 clusters
Round 2, (196, 6)
New Phase, modularity 0.5854071220324866
Iteration #1, 5 vertices Moved, modularity: 0.4840561224489796, cluster#: 3
3 clusters
 {0: 0, 2: 0, 5: 0, 6: 0, 9: 0, 13: 0, 15: 0, 18: 0, 20: 0, 21: 0, 22: 0, 31: 0, 38: 0, 46: 0, 50: 0, 52: 0, 55: 0, 60: 0, 62: 0, 68: 0, 78: 0, 93: 0, 96: 0, 64: 1, 1: 1, 65: 1, 97: 1, 69: 1,
 71: 1, 8: 1, 41: 1, 83: 1, 23: 1, 24: 1, 26: 1, 95: 1, 3: 2, 7: 2, 11: 2, 29: 2, 32: 2, 33: 2, 45: 2, 47: 2, 53: 2, 59: 2, 63: 2, 66: 2, 67: 2, 74: 2, 75: 2, 82: 2, 84: 2, 89: 2, 92: 2, 4: 3,
 10: 3, 17: 3, 19: 3, 25: 3, 30: 3, 34: 3, 35: 3, 36: 3, 37: 3, 40: 3, 43: 3, 48: 3, 51: 3, 56: 3, 57: 3, 70: 3, 72: 3, 76: 3, 77: 3, 79: 3, 81: 3, 85: 3, 86: 3, 87: 3, 91: 3, 98: 3, 99: 3, 39:
 4, 12: 4, 16: 4, 88: 4, 58: 4, 28: 4, 61: 4, 94: 4, 73: 5, 42: 5, 44: 5, 14: 5, 80: 5, 49: 5, 54: 5, 90: 5, 27: 5}
```

Figure 2: Execution of Louvain's Algorithm

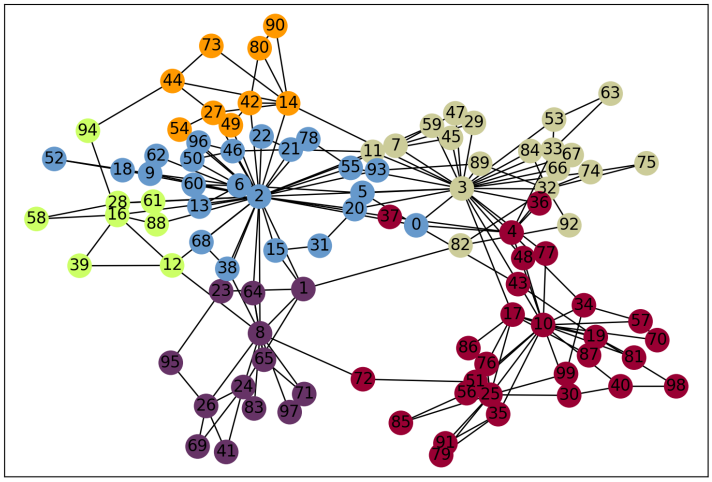

Figure 3: Visualization of Community Detection Results

To reproduce the result, you can simply execute `python main.py`

5 Discussions

Exercise 1 Briefly describe the principle of the community detection algorithm you use.

Solution. Mentioned in Section 3 □

Exercise 2 In addition to visualization, what other applications does the community detection algorithm have?

Solution.

1. Reveal the correlation between nodes. Nodes that are in the same community have more similarities than those that are not. A recommendation system can be developed based on this principle.
2. Given the community structure, we can quickly locate those links or nodes that are out of the community pattern, which may indicate a suspicious event. Some social measures can thus be built based on community detection results to evaluate the outliers.
3. Based on community information and similarity between vertices, we can perform link prediction task for the network.

□