# Lab #3: Community Detection in Networks

EE447 Mobile Network, Luoyi Fu, Spring 2021

**Due**: *Sunday, May 23rd*

Name: Hongjie Fang Student ID:518030910150 Email: galaxies@sjtu.edu.cn

## 1 Purpose and Objective

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.

This Lab focuses on the community detection algorithms for network. In this lab, you will use python to complete the community detection under a specific network and visualize the network based on the results. You need to find dataset on the Internet by yourself, use any community detection algorithm to detect the community, and visualize the network based on the community detection results. After programming, you need to answer two questions listed as follows.

1. Briefly describe the principle of the community detection algorithm you use.
2. In addition to visualization, what other applications does the community detection algorithm have? Please list at least three.

## 2 Algorithms

I implement two network detection algorithms in this lab, namely Louvain Algorithm* and Leiden Algorithm†. They are all based on the notion of modularity. In this section, I will first briefly introduce the definition and meaning of modularity, then introduce two algorithms in details.

### 2.1 Modularity: the Key to the Community

The notion of modularity‡ is first introduced in the field of physics, and is first brought into the community detection area by Louvain algorithm*. The modularity is used to measure whether the division of a community is relatively good result, which has a high similarity of nodes inside the community and a low similarity of nodes outside the community. The formal definition of modularity $Q$ is shown as follows.

$$Q = \frac{1}{m}\sum_{i,j}\left[A_{ij} - \frac{k_i k_j}{2m}\right]\delta(c_i, c_j)$$

where $A_{ij}$ 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, $\delta(c_i, c_j)$ is 1, otherwise $\delta(c_i, c_j)$ is 0.

*Blondel, Vincent D., et al. "Fast unfolding of communities in large networks." Journal of statistical mechanics: theory and experiment 2008.10 (2008): P10008.

†Traag, Vincent A., Ludo Waltman, and Nees Jan Van Eck. "From Louvain to Leiden: guaranteeing well-connected communities." Scientific reports 9.1 (2019): 1-12.

‡Newman, Mark EJ. "Analysis of weighted networks." Physical review E 70.5 (2004): 056131.

In community detection, modularity is regarded as an objective function to optimize. However, the exact modularity optimization is a problem that is computationally hard[‡]. As a result, approximation algorithms are necessary for modularity optimization, which leads to many modularity-based community detection algorithm. Therefore, we say that modularity is the "key" to the community.

## 2.2 Louvain Algorithm

The main idea of Louvain algorithm can be summarize as "**greedily combine the node with the community of its neighbors in order to increase modularity, then reconstruct the graph according to the community, and repeat the process until modularity converges**".

In short, the procedure of Louvain algorithm can be listed as follows.

**Step 1**. Check every node in the graph in a random order, and determine whether combine it with the community of one of its neighbor may increase the modularity. If so, combine the current node with the community which can receive the biggest modularity gain.

**Step 2**. Repeat Step 1 until no modularity gain is available. Then go to Step 3.

**Step 3**. Reconstruct the graph and combine all nodes in a community into one node representing for the community, and the edge between nodes becomes the edge between communities (which may result in self-loop, and it is a valid situation).

**Step 4**. Go back to Step 1 and repeat Step 1 to Step 3, until the reconstructed graph is same as the given graph, *i.e.*, no modularity gain is available under the current circumstances.

An example* of the procedure of Louvain algorithm is illustrated in Fig. 1 as follows.

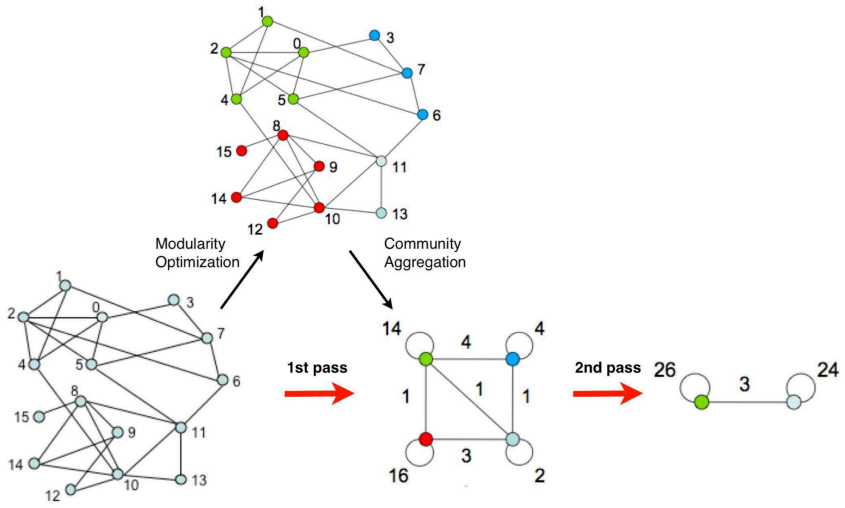


**Fig. 1.** An example of Louvain algorithm

## 2.3 Leiden Algorithm

Leiden algorithm is an improvement of Louvain algorithm. Notice that Louvain algorithm may produce badly-connected communities, that is, a community may be disconnected if we only focus on edges inside the community, which is caused by the graph reconstruction process. The Leiden

*Blondel, Vincent D., et al. "Fast unfolding of communities in large networks." Journal of statistical mechanics: theory and experiment 2008.10 (2008): P10008.

‡Newman, Mark EJ. "Analysis of weighted networks." Physical review E 70.5 (2004): 056131.

algorithm states that **we cannot reconstruct the graph directly according to the their community, but reconstruct the graph by only combine those well-connected subset of communities**. The combination process is simply a copy of the first step of node merging, except that we only merge nodes within the same community and disallow the bad-connected nodes to be merged together. Besides, the Leiden algorithm also use the idea of breadth-first search to optimize the first step of node merging.

In short, the procedure of Leiden algorithm can be listed as follows.

**Step 1**. Initialized a queue with every node in the graph in a random order. For every iteration, fetch a node from the queue and determine whether combine it with the community of one of its neighbor may increase the modularity. If so, combine the current node with the community which can receive the biggest modularity gain. Then, push the neighbor of the changed node into the queue, and we only need to examine those nodes because of the locality of the community modifications. Repeat Step 1 until the queue is empty.

**Step 2**. Refine the partition obtained by Step 1 by only considering the nodes in the partition at one time. Intialize the refined partition as a singleton partition, then merge the nodes into the community in the refined partition if the community is well-connected. Repeat the process until no merge is available.

**Step 3**. Reconstruct the graph and combine all nodes in a community into one node representing for the community according to the refine dcommunity, and the edge between nodes becomes the edge between communities (which may result in self-loop, and it is a valid situation).

**Step 4**. Go back to Step 1 and repeat Step 1 to Step 3, until the reconstructed graph is same as the given graph, *i.e.*, no modularity gain is available under the current circumstances.

An example[†] of the procedure of Leiden algorithm is illustrated in Fig. 2 as follows.

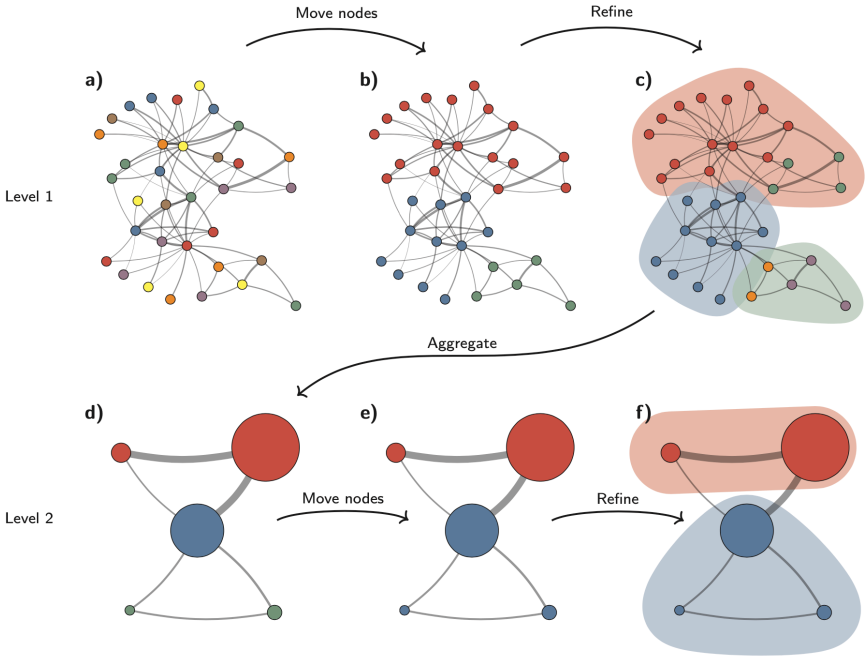


**Fig. 2.** An example of Leiden algorithm

[†]Traag, Vincent A., Ludo Waltman, and Nees Jan Van Eck. "From Louvain to Leiden: guaranteeing well-connected communities." Scientific reports 9.1 (2019): 1-12.

# 3 Implementations and Experiments

Notice that I do not use any advanced clustring or community detection API in this lab in order to deepen the understanding of the community detection algorithms mentioned before. I implement the Louvain algorithm and the Leiden algorithm from zero. The process is not so difficult since the algorithms are more about the exquisite thoughts but less about the difficult techniques.

For dataset, I use two datasets, namely the DBLP collabration dataset[§] and the Starwars social network dataset[¶]. The first dataset is relatively large, which is used to test the speed and performance of two community detection algorithms. The second dataset is relatively small, which is used to visualize the results of the community detection in an intuitive way.

## 3.1 Speed and Performance: DBLP Collabration Dataset

The Leiden algorithm claims that it runs up to 20x faster than the Louvain algorithm, and it produces less bad-connected communities. Unfortunately, the quality of the community detection CANNOT only be determined by the modularity, and high modularity can also produces bad-connected communities. Therefore, Leiden algorithm may produce results with lower modularity but higher quality. We now test the speed and performance of two algorithms on the DBLP Collabration dataset, since it has more than 300, 000 nodes and more than 1,000,000 edges, which is a relatively great benchmark to test the speed and performance of a community detection algorithm.

Now, we list the quantitative results in Tab. 1 as follows.

| Algorithms | Epoch 1 | Epoch 2 | Epoch 3 | Epoch 4 | Epoch 5 | Total Time |
|---|---|---|---|---|---|---|
| Louvain | 200.84 | 92.32 | **37.13** | **6.58** | **1.07** | 337.94 |
| Leiden | **143.71** | **77.51** | 62.13 | 32.43 | 13.36 | **329.14** |

**Tab. 1.** The speed of Louvain algorithm and Leiden algorithm in each epoch (unit: second)

From Tab. 1, we can see that in the first two epochs, the Leiden algorithms is obviously faster than Louvain algorithm, but in the next few epochs, Leiden algorithm is slower than Louvain algorithm. It is because that the Leiden algorithm does not reconstruct the graph thoroughly, and in the next epochs, the size of the induced graphs in Louvain algorithm and Leiden algorithm may have a huge difference, resulting in the speed difference. At least in the first epoch, the size of the graph is same in two algorithms, so according to the experimental results on DBLP collabration dataset, we can conclude that when the size of the graph is fixed, Leiden algorithm is faster than Louvain algorithm.

The Louvain algorithm gives a community partitioning result of modularity about 82.14, and the Leiden algorithm gives a community partitioning result of modularity about 68.07. The reason of the modularity difference is already explained in the first paragraph. According to the number of communities in the partition, the Leiden algorithm produces about 5 times more communities than the Louvain algorithm, which implies that Leiden algorithm produces less bad-connected communities indirectly.

In all, **the Leiden algorithm is faster than the Louvain algorithm**, and **the Leiden algorithm has lower modularity score, but produces less bad-connected communities**.

## 3.2 Visualization: the Starwars Social Network Dataset

We use Louvain algorithm and Leiden algorithm to find the communities in the Starwars social network dataset, and the results are illustrated as follows.

[§]J. Yang and J. Leskovec. Defining and Evaluating Network Communities based on Ground-truth. ICDM, 2012

[¶]https://www.kaggle.com/ruchi798/star-wars

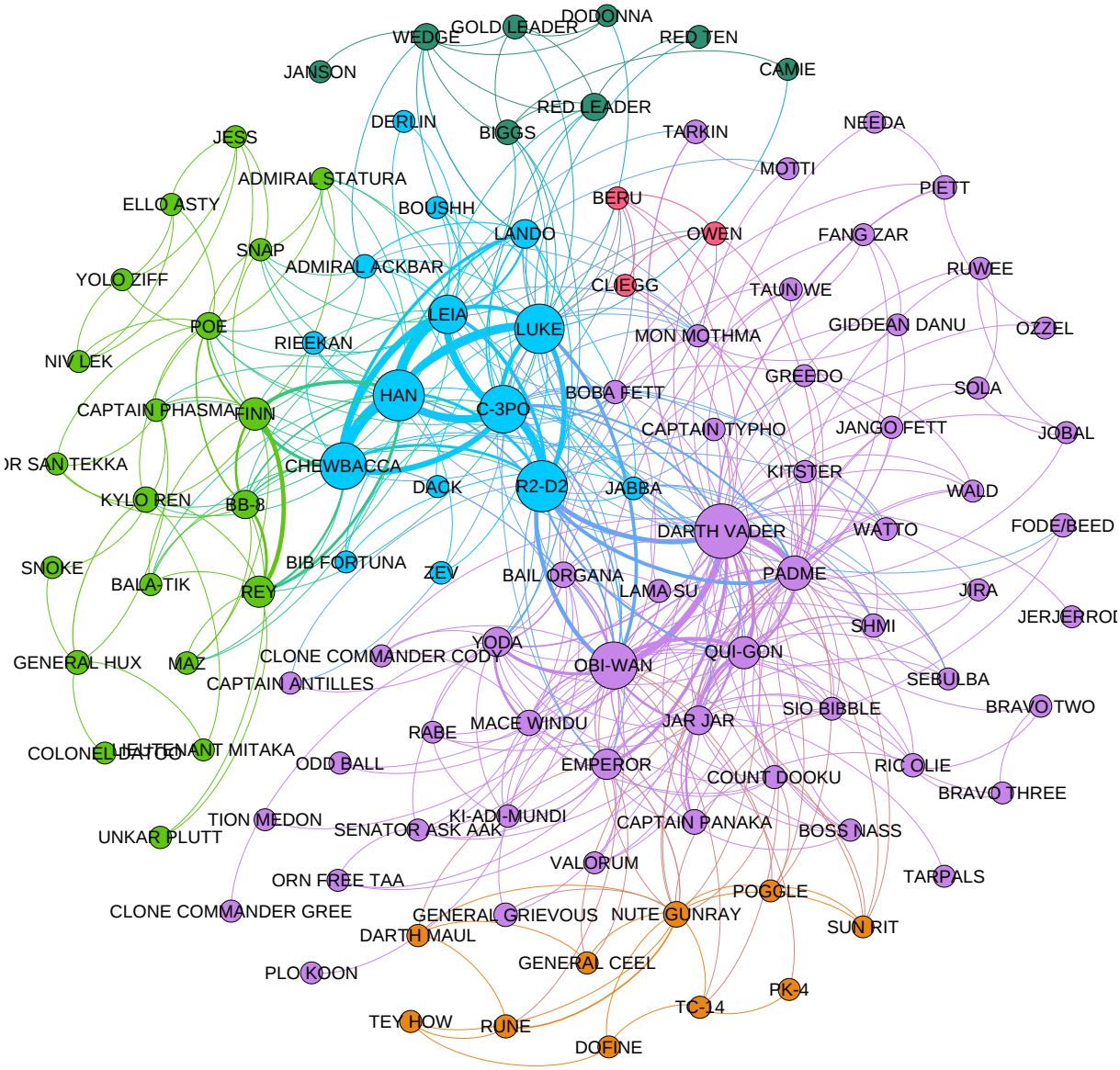


**Fig. 3.** The visualization result of the community detection results of Louvain algorithm on Starwars social network dataset, where different colors indicates different communities

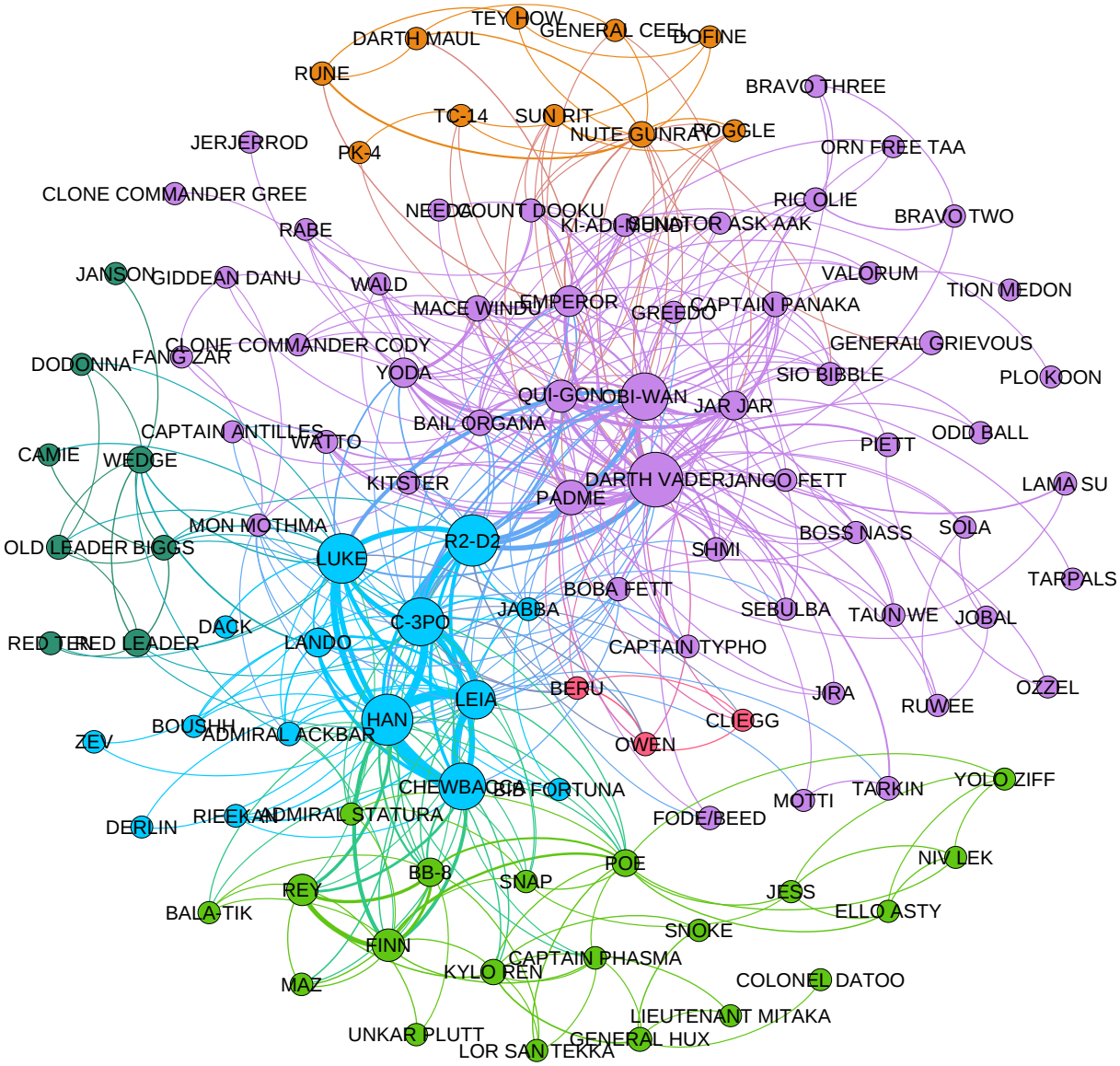


**Fig. 4.** The visualization result of the community detection results of Leiden algorithm on Starwars social network dataset, where different colors indicates different communities

From Fig. 3 and Fig. 4 we can observe that both Louvain algorithm and Leiden algorithm performs well on small datasets. According to the knowledge of Starwars series movies, we can see that both algorithms have recognized the Galatic Empire community (the purple colored communities in both figures), and the Rebel Alliance (the blue colored communities in both figures). Therefore, we can conclude that both algorithms can find out communities and achieve a high accuracy.

# 4 Further Discussions

In this section, we will answer the questions mentioned in the first section.

1. **Q**: Briefly describe the principle of the community detection algorithm you use.

   **A**: Both Louvain algorithm and Leiden algorithm is based on the principle that **greedily merge the nodes into the community of its neighbors to increase modularity**. But instead of only focusing on the modularity, Leiden algorithm also considers the problem of bad-connected communities, and use refined graph to improve the quality of partition.

2. **Q**: In addition to visualization, what other applications does the community detection algorithm have? Please list at least three.

   **A**: Other applications of community detection includes:

   - **Customized Recommendations** (Smart Advertising and Targeted Marketing): find the communities in the user group, and use other users' habits to make recommendations to the user in the same community.
   - **Link Prediction**: use community detection results to predict whether there is a link between two nodes. If two nodes are in the same community, then the probability that there is an edge between them is a little bit higher. Therefore, the community detection algorithms can be used as an auxiliary approach to link prediction.
   - **Network Privacy**: Community detection provides a group-level point of view of a network, and it can also be used to break privacy of people on weak signal networks like Bitcoins, which leads to some applications in the weak signal networks to extract the privacy of users.
   - **Criminology**[‖]: use community detection to find out the criminal groups and prevent further criminal activities.
   - **Politics**[‖]: politicians can use the community detection results to track their supporters and fight for the votes of other peoples in the same community with their supporters.

# 5 Conclusion

In this lab, we dive deeply into the community detection algorithms. I implement two famous modularity-based algorithms by myself, namely the Louvain algorithm and the Leiden algorithm. Their ideas are quite simple: try to optimize the modularity by greedily merge singleton nodes into its neighborhood communities. Such simple idea may lead to satisfactory results illustrated before, which suprised me and make me to think that the simplicity of the algorithm is also an important view to judge the quality of an algorithm. The algorithms also inspires me to develop more simple but efficient algorithms in the further researches.

[‖]Karataş, Arzum, and Serap Şahin. "Application areas of community detection: A review." 2018 International congress on big data, deep learning and fighting cyber terrorism (IBIGDELFT). Ieee, 2018.

In order to finish the lab, I read many papers about community detection, and gain a lot of knowledge about the community detection fields, which helps me a lot in the data mining area. Also, in order to completes visualization, I learnt to use Gephi** to visualize the network in a more intuitive way, which also benifits me in the further researches.

The full implementation codes of the lab is available in my github repository, or you can see the attached files for details. To execute the code, just enter the `community` folder and execute one of the followings:

```
python louvain_dblp.py
python leiden_dblp.py
python louvain_starwars.py
python leiden_starwars.py
```

The four lines correspond to the two algorithms on two different datasets introduced before. For other questions about the source code, please feel free to send an e-mail†† to me.

**https://gephi.org/

††mailto:galaxies@sjtu.edu.cn