Graph Theory Applied to Biological Network Alignment

Kai Yi
Institute of Artificial Intelligence and Robotics, Xian Jiaotong University, Xian, Shaanxi, P.R.China
National Engineering Laboratory for Visual Information
Processing and Applications, Xi'an Jiaotong University, Xi'an, Shaanxi, P.R.China
yikai2015@stu.xjtu.edu.cn

Abstract

Graph theories are popular among a lot of fields. In this paper, we will review their usages on solving the problem of biological network alignment. Due to the fact that this paper is exactly the solution of the open question from Professor Hayes' homepage, after reviewing those mentioned above, we will mainly talk about the design principle, fundamental ideas and experiments of the task. In order to show our results more friendly, I visualized them by using Python and Mathematica respectively.

1. Introduction

Graph theories has been widely used in a wide range of fields such as biology [1] and social networks [2]. In this article, we will review the basic ideas of graph theory, and it's applications on the task of biological network alignment.

Graphs are used to represent the topological relationship of an enormous array [3]. A graph G(V, E) is the gather of plenty of nodes and edges with a complex way interacting among. V represents the set of nodes while E represents the set of edges usually.

A networks can be seen as a graph. Network alignment is aimed at comparing the similarity of two graphs. One problem similar to network alignment is called subgraph isomorphism, which asks whether graph M is a subgraph of N. However, this is a NP-complete problem, which means that no non-binomial solution can be found [4], [5]. Network alignment is as difficult as this problem. Besides, in the context of biology, the network is easily to be affected by random noise such as missing edges, false edges or both of them [6]. Due to biological variation, which means that one or more the cases mentioned above occurs, it is not very evident how to judge the rate of fitness.

In order to assess the similarity of two graphs, one popular method is to calculate the percentage of aligned edges, which is called *edge correctness* [1] (EC). However, two different alignments may have similar ECs. Further, it is not evident how to use EC to guide an alignment algorithm as maximizing EC is an NP-hard problem. That is, if it can be handled, the subgraph isomorphism problem can be solved as well.

There are plenty of efforts trying to solve the network alignment problem. They can roughly classified into two mainstream methods: local network alignment and global network alignment. The basic idea of the formal method is to compare the similarity of each local subregion of two selected networks independently instead of the whole. And global network alignment is aimed at maximizing the overall match between two networks by regarding the network as a whole.

So far, compared to the global network alignment method, that based on the local alignment is the majority [7], [8]. [7] proposed the PathBLAST, which searches for high-scoring pathway alignments between two networks. It considered not only the homology between the aligned proteins but also the probabilities that Proton Pump Inhibitors (PPIs) in the path are true PPIs and not false-positives. GRAEMLIN, proposed by [9], is the first method that can identify dense conserved subnetworks of arbitrary structure. What's more, it can score a module by computing the log-ratio of the probability that the module is subject to evolutionary constraints and the probability that the module is under no constraints while considering phylogenetic relationships between species whose networks are being aligned. Global network alignment is another very active method [10], [11]. ISORANK proposed by [10] used the heuristic that two nodes are a good match if their respective neighbours also match well. This assumption based on spectral graph theory greatly improved the scores of aligning pairs of

Table 1. Total degrees of graphs with different numbers of nodes.

Nodes	Edges(Calculated)	Edges (Simple Graph)
10	100	60
100	99	99
1000	1999	1995
10000	32999	32992
573	1202	1202

nodes from different networks. Further, [11] proposed GRAEMLIN by referring to the known network alignment set. They developed a learning algorithm that uses a training set of known network alignments and their phylogenetic relationships to learn parameters for its scoring function. Besides, it can adapt the learned objective function to any set of networks automatically.

2. Implementation

In this section, we will talk about all the information concerning the question proposed by Professor Hayes as clear as possible. Firstly, I will review the definition of the problem. Then talk about my configure of my operating system and coding environments. Finally, I will introduce the design principle of my implementation.

2.1. Question Definition

According to the guidance from the homepage of Professor Hayes, the question is defined as following:

You are Give a text file representing a network. The first line of the file is N, the number of nodes. You will name the nodes from 0 through N-1. The remaining lines will have two integers per line, representing an edge. You don't know in advance how may edges there are, you just keep reading until you reach end of file. When you are done, you are to compute the number of CONNECTED COMPONENTS in the graph, and ourput a single integer. In addition, in your write up, include a histogram of the distribution of DEGREES of nodes. That is, how many nodes have degree zero, degree 1, etc., up to the max degree. The graphs can not only be undirected graph but also be directed graph. If it's an undirected graph, you need to do those mentioned above. In the case of directed graph, you also need to enumerate the number of strongly, and weakly connected components.

We modified the original question slightly. Although I'm an undergraduate student, I think it is an opportunity to challenge myself. So I take the extra work for graduate students into consideration. Fortunately, I've overcome several difficulties and solved the extra work successfully.

2.2. Prerequisitory

My operating system is Ubuntu 16.04 LTS. My working environment is Python 3.6, Anaconda, OpenCV 3, Mathematica 11.0 and Visual Studio Code.

2.3. Design Principle

Network alignment like a NP-hard problem, but the problem that I need to solve is far more easier than it. In some way, it can be roughly categorized into a typical edge-counting problem in a graph.

The basic workflow of my implementation is to read the given .txt file firstly. And then classifies it into two parts, the first part is the first line, which represents how many nodes are in the given graph, and the second part is the rest content, which means the edges among nodes. I just regard the graph as a n-D array and store the relationships (edges) of different nodes by using a 2-D array. Finally, calculating the degree of every node and plot the distribution of them.

As the graph can be both undirected and directed, there are different handling of those lines except the first line. For the undirected condition, I implemented my idea by using python and Mathematica respectively. For the directed one, I represented my Mathematica code.

3. Experiments and Results

The provided dataset consists of five data parts with different number of nodes including nodes = 10, 100, 1000, 10000 and 573 (it consists of several subgraphs). The result of total degrees of those graphs mentioned above is in the following Table 1:

As for the visualization of the directed graphs, simplified directed graphs, distribution of degree, undirected graphs, and subgraphs can refer to the Figures in Appendix.

4. Conclusion

In this article, we reviewed the fundamental theories of graph theory and network alignment. Then talked about the usage of graph theory to solve network alignment problems. Further, I reclaimed the design principle, fundamental ideas and experiments of the task put forward by Professor Hayes. The visualization of different graphs represented the topological structure of different graphs, which really motivated my further study towards to this field.

Acknowledgement

Thanks for the help of Professor Wayne Hayes for providing teh aimed problem and kind instructions.

References

[1] Oleksii Kuchaiev, Tijana Milenković, Vesna Memišević, Wayne Hayes, and Nataša Pržulj. Topological network alignment uncovers biological function and phylogeny. *Journal of the Royal Society Interface*, page rsif20100063, 2010.

[2] John Scott. *Social network analysis*. Sage, 2017.

[3] Vittoria Colizza, Alessandro Flammini, M Angeles Serrano, and Alessandro Vespignani. Detecting rich-club ordering in complex networks. *Nature physics*, 2(2):110, 2006.

[4] Stephen A Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158. ACM, 1971.

[5] Ron Shamir and Dekel Tsur. Faster subtree isomorphism. *Journal of Algorithms*, 33(2):267–280, 1999.

[6] Kavitha Venkatesan, Jean-Francois Rual, Alexei Vazquez, Ulrich Stelzl, Irma Lemmens, Tomoko Hirozane-Kishikawa, Tong Hao, Martina Zenkner, Xiaofeng Xin, Kwang-Il Goh, et al. An empirical framework for binary interactome mapping. *Nature methods*, 6(1):83, 2009.

[7] Brian P Kelley, Bingbing Yuan, Fran Lewitter, Roded Sharan, Brent R Stockwell, and Trey Ideker. Pathblast: a tool for alignment of protein interaction networks. *Nucleic acids research*, 32(suppl_2):W83–W88, 2004.

[8] Zhi Liang, Meng Xu, Maikun Teng, and Liwen Niu. Netalign: a web-based tool for comparison of protein interaction networks. *Bioinformatics*, 22(17):2175–2177, 2006.

[9] Jason Flannick, Antal Novak, Balaji S Srinivasan, Harley H McAdams, and Serafim Batzoglou. Graemlin: general and robust alignment of multiple large interaction networks. *Genome research*, 16(9):1169–1181, 2006.

[10] Rohit Singh, Jinbo Xu, and Bonnie Berger. Pairwise global alignment of protein interaction networks by matching neighborhood topology. In *Annual International Conference on Research in Computational Molecular Biology*, pages 16–31. Springer, 2007.

[11] Jason Flannick, Antal Novak, Chuong B. Do, Balaji S. Srinivasan, and Serafim Batzoglou. Automatic parameter learning for multiple network alignment. In *International Conference on Research in Computational Molecular Biology*, pages 214–231, 2008.

Appendix

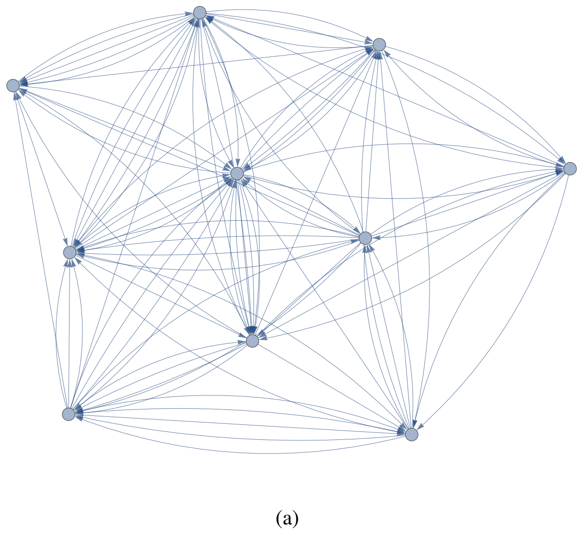
4.1. Source Code of Implementation by Using Mathematica

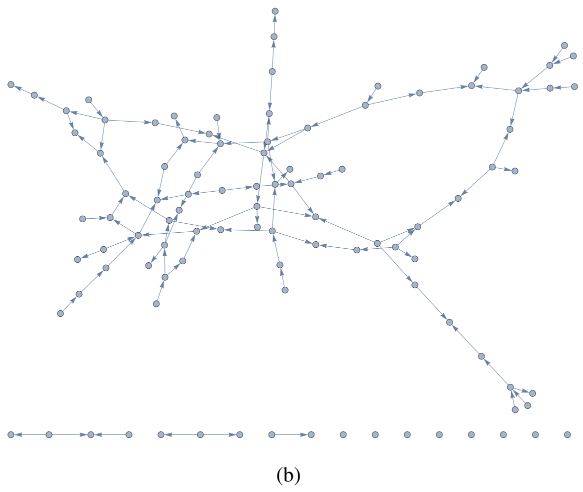
```
Fun[csvpath_, number_, exportpathbase_ ] :=
Module[{csvdata, dgraph, simpdgraph, ugraph},
    csvdata = Import[csvpath];
    dgraph =
    Graph[Range[0, number - 1], (#[[1]] \[DirectedEdge] #[[2]]) & /@
    csvdata];
    simpdgraph = SimpleGraph[dgraph];
    ugraph =
    Graph[Range[0, number - 1], (#[[1]] <-> #[[2]]) & /@ csvdata];

    Export[exportpathbase <> ToString[number] <> "_histrogram.pdf",
    Histogram[VertexDegree[simpdgraph], {1},
    PlotTheme -> "Scientific"]];
    Export[exportpathbase <> ToString[number] <> "_dgraph.pdf",
    simpdgraph // Framed];
    Export[exportpathbase <> ToString[number] <> "_dgraph_r.pdf",
    Rasterize[simpdgraph // Framed, RasterSize -> 1000]];
    Export[exportpathbase <> ToString[number] <> "_ugraph.pdf",
    ugraph // Framed];
    Export[exportpathbase <> ToString[number] <> "_ugraph_r.pdf",
    Rasterize[ugraph // Framed, RasterSize -> 1000]];
    ];

Fun["/home/kyi/Desktop/graph-alignment/data/s1.csv", 573, \
"/home/kyi/Desktop/graph-alignment/data/"]
```

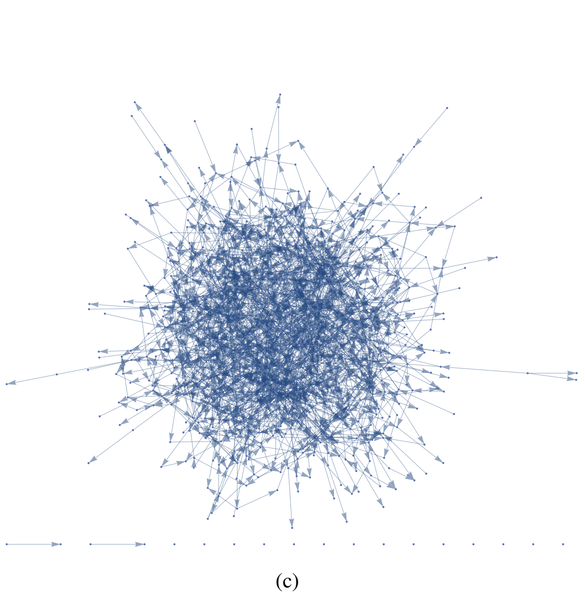
4.2. Source Code of Implementation by Using Python

```
# Demonstrate the Undirected Graph Alignment
# Created by Kai Yi

from __future__ import print_function
from __future__ import division

import numpy as np
import math
import cv2
import matplotlib.pyplot as plt

file_dir = "/home/kyi/Desktop/graph-alignment/data/n10000.txt"
first_line = True
count = 0

f = open(file_dir, 'r')

elements = f.read().split()
total_nodes = int(elements[0])
store = np.zeros((total_nodes, total_nodes), dtype=np.int)
degrees = np.zeros(total_nodes, dtype=np.int)
mhistogram = np.zeros(total_nodes, dtype=np.int)
```

```python
print(len(elements))

for i in range(1, len(elements)-1):
    store[int(elements[i])][int(elements[i+1])] += 1
    i += 2

# Print the Histogram
for i in range(0, total_nodes):
    for j in range(0, total_nodes):
        if store[i][j] != 0:
            degrees[i] += 1
        else:
            continue
print(degrees)

for i in range(0, len(degrees)):
    mhistogram[degrees[i]-1] += 1
print(mhistogram)

plt.bar(range(math.log10(total_nodes)), mhistogram)
# plt.bar(range(20), mhistogram)
plt.show()
```


(a)


(b)


(c)

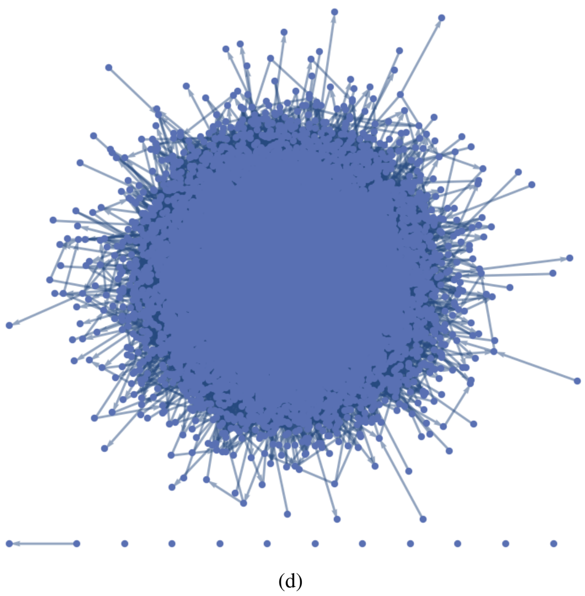

(d)

Figure 1. (a), (b), (c), (d) are four visualization of directed graphs for nodes = 10, 100, 1000, 10000 respectively.

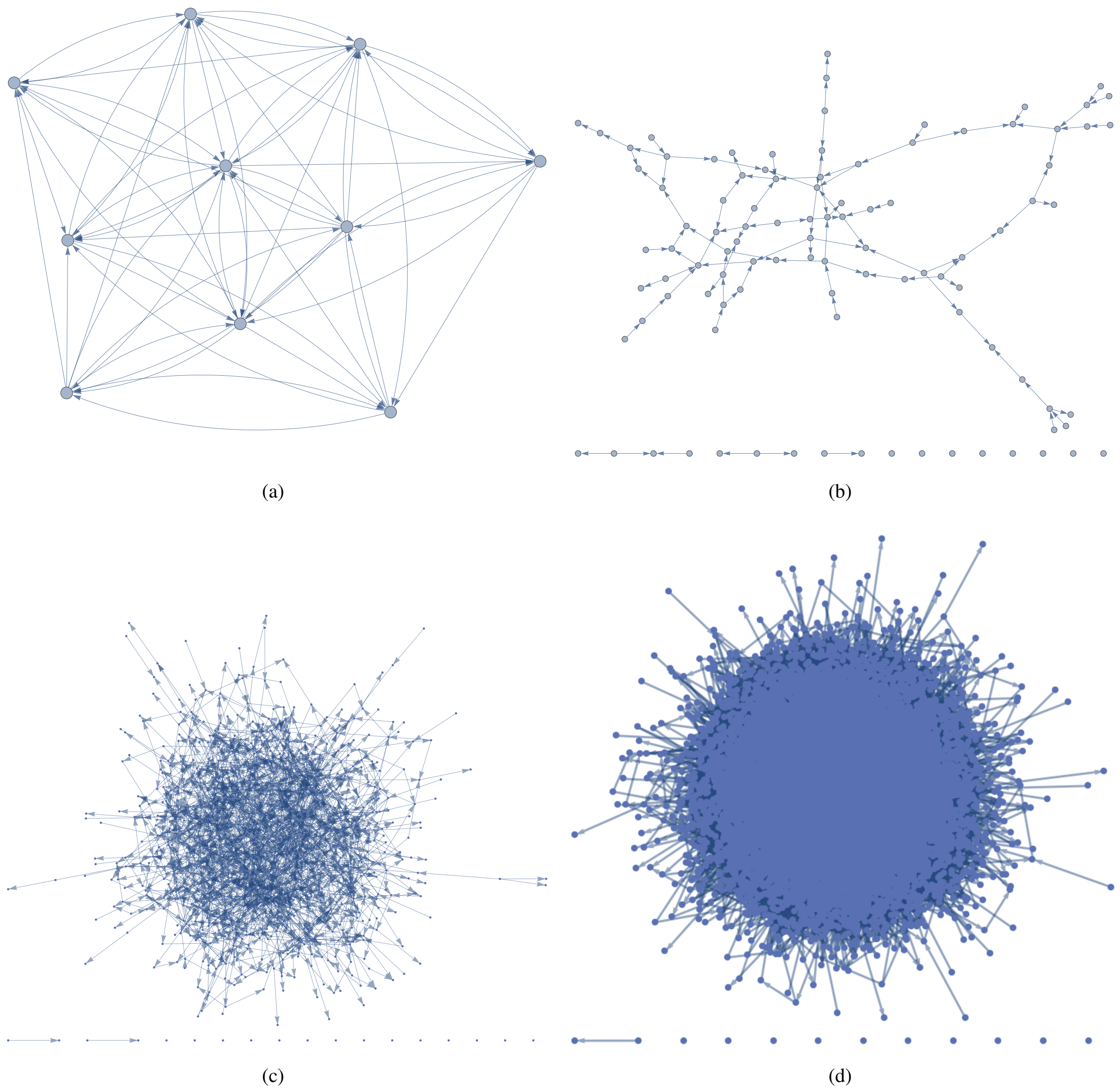


Figure 2. (a), (b), (c), (d) are four visualization of simplified directed graphs for nodes = 10, 100, 1000, 10000 respectively.

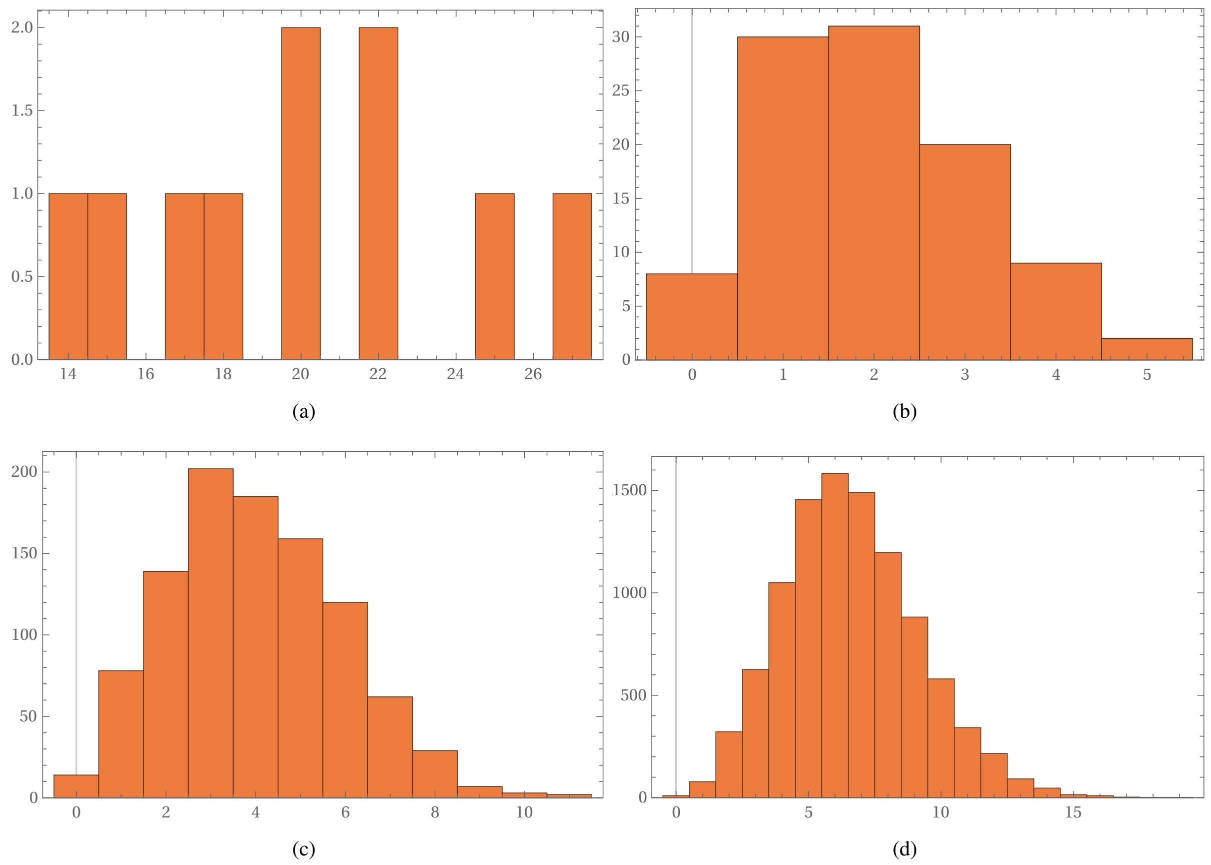
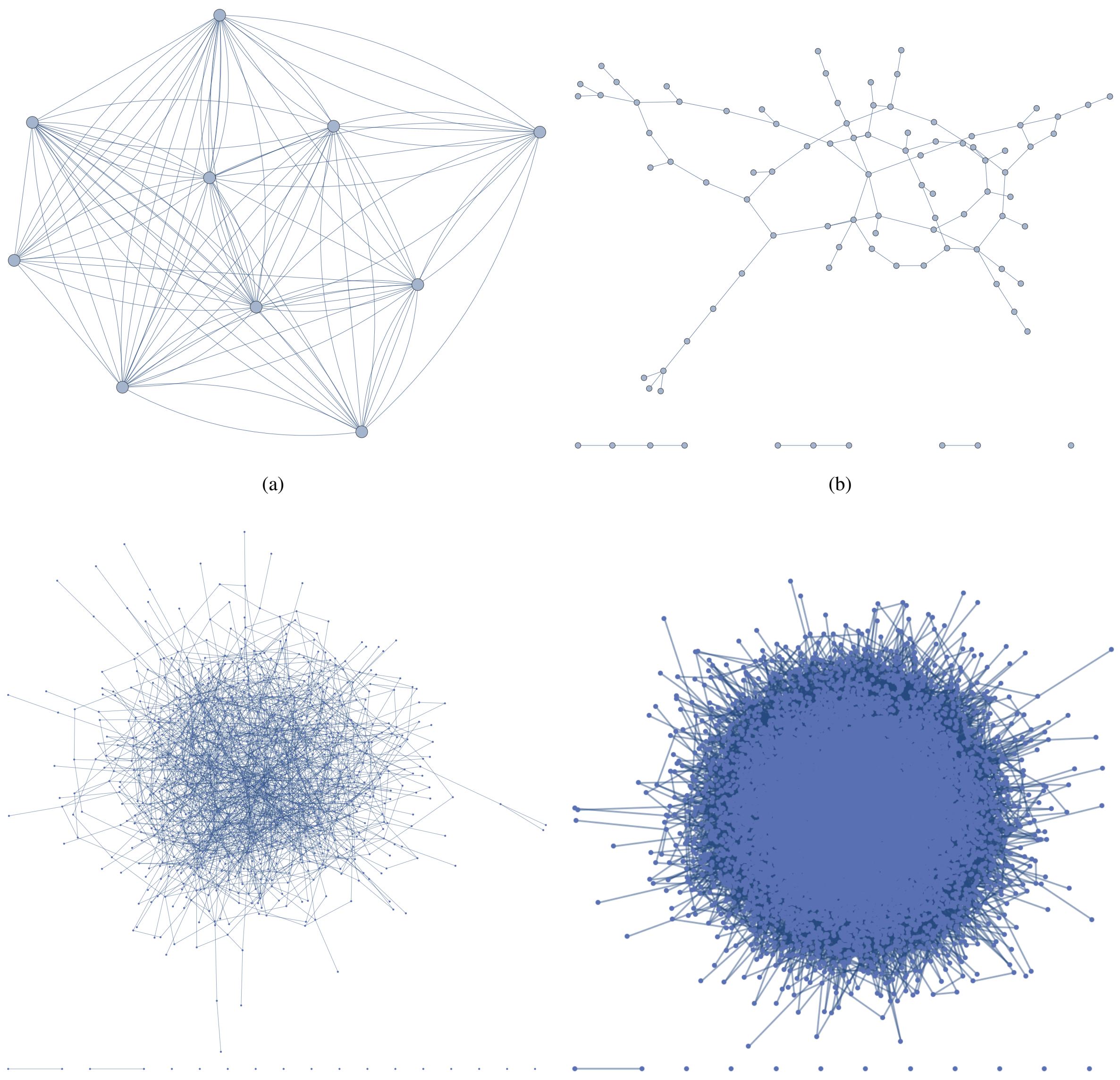
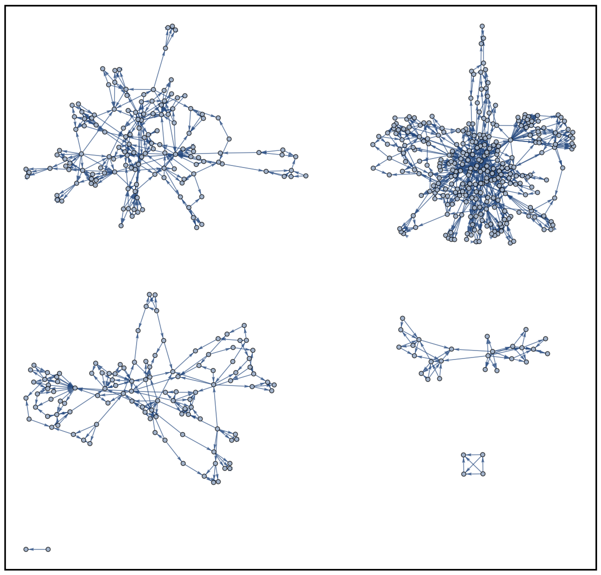


Figure 3. The visualization of the distributions of degrees. (a), (b), (c), (d) are the distributions of the degrees for nodes = 10, 100, 1000, 10000 respectively.

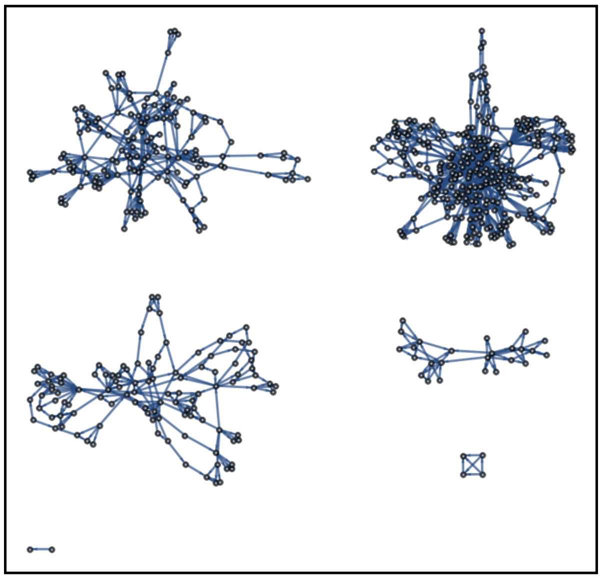


(a) (b) (c) (d)

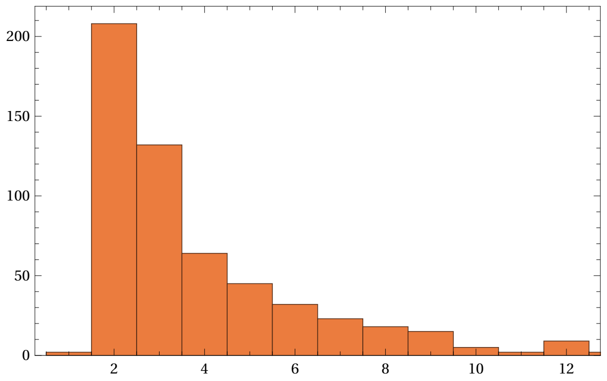
Figure 4. The visualization of the distributions for nodes with variant degrees. (a), (b), (c), (d) are the undirected graphs for nodes = 10, 100, 1000, 10000 respectively.



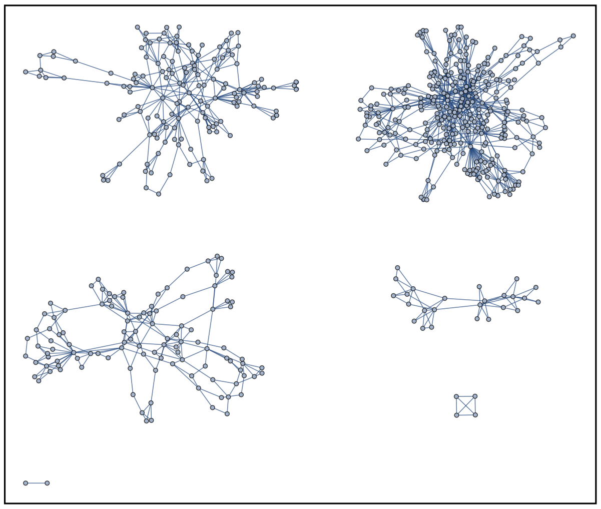
(a)



(b)



(c)



(d)

Figure 5. As the graph in dataset ‘s1’ is composed several subgraphs, I picked it up from those mentioned above. The undirected subgraph, simplified undirected subgraph, distribution of degree, and directed subgraph is in this figure.