Complex Dynamic Networks Project

by

Aylin Naebzadeh

(99522185)

Advanced Big Data Analysis Laboratory

Department of Computer Engineering

IRAN UNIVERSITY OF SCIENCE AND TECHNOLOGY

June, 2025

Contents

List of Figures

List of Tables

Chapter 1

Input File

1.1 Input File

For this problem we were given two excel files. The first one contains all the nodes with their labels. Each label can have a value between L1 to L7 or Unknown. The second file contains all the edges of the network. Each edge has an start node and an end node. Based on the descriptions, our network is directed.

1.2 Technologies

- Python programming language (Modules and predefined libraries such as NetworkX, SKlearn, etc.)
- Google Colab platform
- Django (for web implementation)
- React (for web implementation)
- Visual Studio Code

```
import pandas as pd
import networkx as nx
from pprint import pprint
import collections
import gravis as gv
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.colors import rgb2hex
import networkx.algorithms.isomorphism as iso
import itertools
from networkx.algorithms import community
import EoN
from collections import defaultdict
import random
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree
import csv
```

Chapter 2

Problem Description

2.1 First Step

The first step for analysing the graph is to read its edges and nodes from the excel files. But we must also consider that the given network is directed, and the direction of edges is important. We must also store the label of each node, since in further analysis it would be used.

All of these steps are shown in the below code snippet.

```
data = pd.read_excel('edges.xlsx')
G = nx.from_pandas_edgelist(data,
            'sourceNodeId', 'targetNodeId',
            create_using=nx.DiGraph())
labels_df = pd.read_excel('nodes.xlsx')
labels_dict = labels_df.set_index('NodeId')['Labels']
                        .to_dict()
nx.set_node_attributes(G, labels_dict, 'label')
```

Now if we `print` the result, we can see that `G` is a network with **2708** nodes and **10556** edges.

We can also visualize the network using the below command to see its structure and have a better sense about our network.

```
gv.d3(G, edge_size_data_source='weight',
   use_edge_size_normalization=True)
```

Here is its picture 2.1.

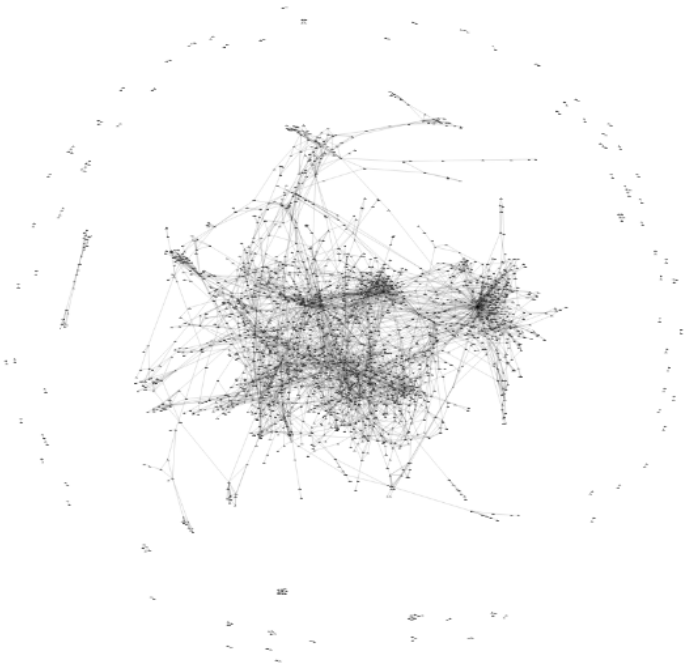


FIGURE 2.1: Visualize the network

2.2 Second Step(General Statistical Information)

I have also calculated some general metrics about the network, which are mentioned in the table 2.1. Besides those general metrics, I have also plot the degree distribution histogram. Here is its picture 2.2.

TABLE 2.1: General Statistical Information About Network.

Metric	**Output**
Diameter	19
Density	0.0014
Degree Centralization	0.06044
Average Shortest Path	6.307544
Average Degree	3.89808
Assortiativity	-0.065871
Transitivity	0.093497

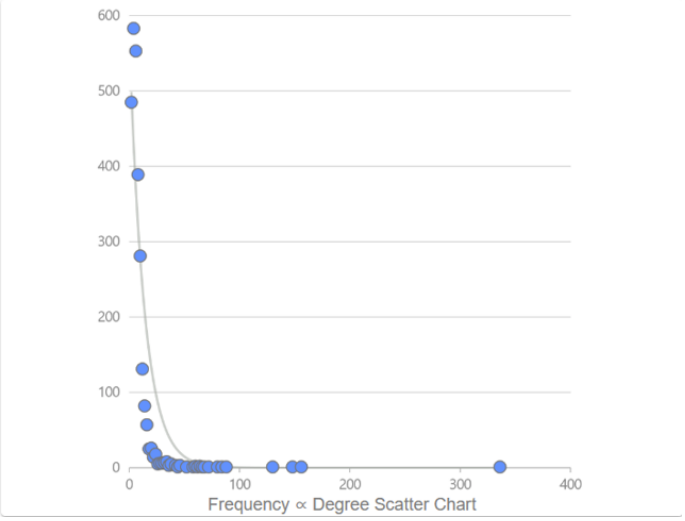


FIGURE 2.2: Degree Distribution Plot

I have also detected the communities using **Greedy Modularity** algorithm, and then set a different color to each community. Here is its picture 2.3.

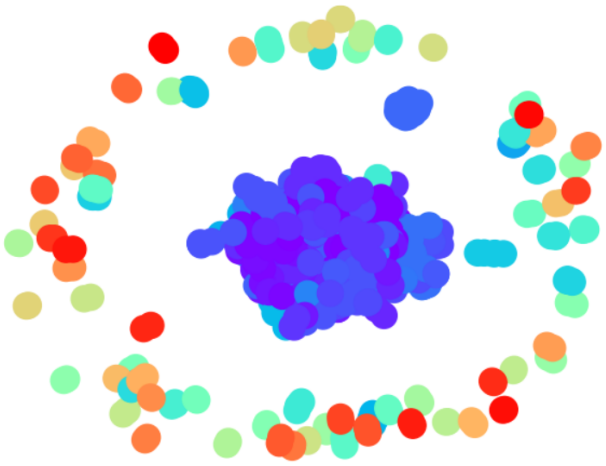


FIGURE 2.3: Communities with Colorization

There are several methods for calculating the centrality of the nodes in the graph, and I have calculated the centrality using four methods.

- Degree Centrality

  Degree centrality is calculated as the number of edges connected to a node, normalized by the maximum possible degree in the graph. In other words, it measures how many neighbors a node has.

  In this network, the top 5 nodes by degree centrality are nodes 35, 6213, 1365, 3229, and 910. These nodes have the highest number of edges connected to them, which means they have the most neighbors in the network.

  One possible interpretation of this result is that these nodes may be important or influential in the network due to their high connectivity. They may act as hubs or bridges between different parts of the network.

- Closeness Centrality

  Closeness centrality is calculated as the reciprocal of the average shortest path length from a node to all other nodes in the graph. In other words, it measures how close a node is to all other nodes in the network.

  In this network, the top 5 nodes by closeness centrality are nodes 35, 6213, 3229, 887, and 4584. These nodes have the highest closeness centrality values, which means they have the shortest average distance to all other nodes in the network.

  One possible interpretation of this result is that these nodes may be important or influential in the network due to their central position. They may be able to quickly disseminate information or influence to other parts of the network due to their short average distance to all other nodes.

- Betweenness Centrality

  Betweenness centrality is calculated as the fraction of all shortest paths between pairs of nodes in the graph that pass through a given node. In other words, it measures how often a node acts as a bridge along the shortest path between two other nodes.

  In this network, the top 5 nodes by betweenness centrality are nodes 35, 3229, 4330, 1365, and 6213. These nodes have the highest betweenness centrality values, which means they are on the highest fraction of shortest paths between pairs of nodes in the network.

  One possible interpretation of this result is that these nodes may be important or influential in the network due to their position as bridges between different parts of the network. They may play a crucial role in connecting different communities or clusters within the network and facilitating the flow of information or influence between them.

- Eigenvector Centrality

  Eigenvector centrality is calculated as the relative importance of a node based on its connections to other important nodes in the network. In other words, it measures how well-connected a node is to other well-connected nodes.

  In your case, the top 5 nodes by eigenvector centrality are nodes 35, 82920, 85352, 210871, and 887. These nodes have the highest eigenvector centrality values, which means they are well-connected to other well-connected nodes in the network.

  One possible interpretation of this result is that these nodes may be important or influential in the network due to their connections to other important nodes. They may be part of a core group of well-connected nodes that play a central role in the network.

The 5 top nodes based on each of these methods is listed in the table 2.2.

TABLE 2.2: Important Nodes Based on Centrality Measures.

Centrality Method	**1st**	**2nd**	**3rd**	**4th**	**5th**
Degree Centrality	35 0.12412264499445881	6213 0.05762837089028445	1365 0.054673069818987806	3229 0.04802364240857037	910 0.03250831178426302
Closeness Centrality	35 0.22276881556604902	6213 0.2211907346794579	3229 0.21982549145258112	887 0.21601312745183981	4584 0.21595173101580423
Betweenness Centrality	35 0.23248831450559276	3229 0.12610085690638292	4330 0.08934413819511756	1365 0.08534091150396639	6213 0.07637499735733927
Eigenvector Centrality	35 (0.6542996956637285)	82920 (0.11789362473669018)	85352 (0.09924151614685083)	210871 (0.0918477470705072)	887 (0.09140594357270748)

2.3 Third Step

Based on the previous step, and the results in table 2.2, which I have found the 5 most important nodes based on different centrality measures, we can claim that node 35 has important rules in our network.

I have also listed 5 nodes which belong to the most number of cliques in the network. You can see the result in table 2.3.

TABLE 2.3: Important Nodes Based on Cliques Count.

Node	**#Cliques**
35	345
6213	200
1365	93
3229	85

Another approach, which I have implemented to find 5 important nodes was community centrality. You can see the result in table 2.4. Based on these results, we

TABLE 2.4: Important Nodes Based on Community Centrality.

Node	Community Centrality
35	0.12412264499445881
103515	0.008127077946065755
103482	0.004432951606944957
1033	0.003694126339120798

can again deduce that the node 35 is the most important node of our network.

2.4 Fourth Step

Since each node has a label, and each label can have a value equal to **L1 to L7** or **Uknown**, I set and define a different color for each of these labels, and then apply them to the network. Here is its picture 2.4.

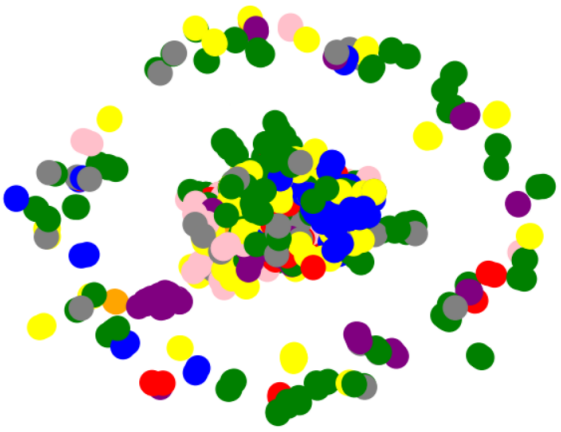


FIGURE 2.4: Colorize the Network Based on Labels.

A) I have also calculated the average clustering coefficient for nodes with the same label. You can see the result in table 2.5. The label values with the highest average

TABLE 2.5: Average Clustering Coefficient for Each Label.

Label	**Average Clustering Coefficient**
L2	0.28941920411679306
L3	0.22783191369053876
L7	0.19980969588910324
L5	0.23239806159978937
L1	0.21999670134686422
L4	0.2667809389989461
L6	0.26910453505399307
Unknown	0.21877637130801686

clustering coefficients are L2 and L6. This suggests that nodes with these label values tend to cluster together more than nodes with other label values. The label values with the lowest average clustering coefficients are L7 and Unknown. This suggests that nodes with these label values tend to cluster together less than nodes with other label values.

B) One way to check if nodes with different label values are evenly distributed throughout a network is to calculate the shortest path length between all pairs of nodes and then compare the distribution of shortest path lengths for pairs of nodes with the same label value and for pairs of nodes with different label values. You can see the result in table 2.6.

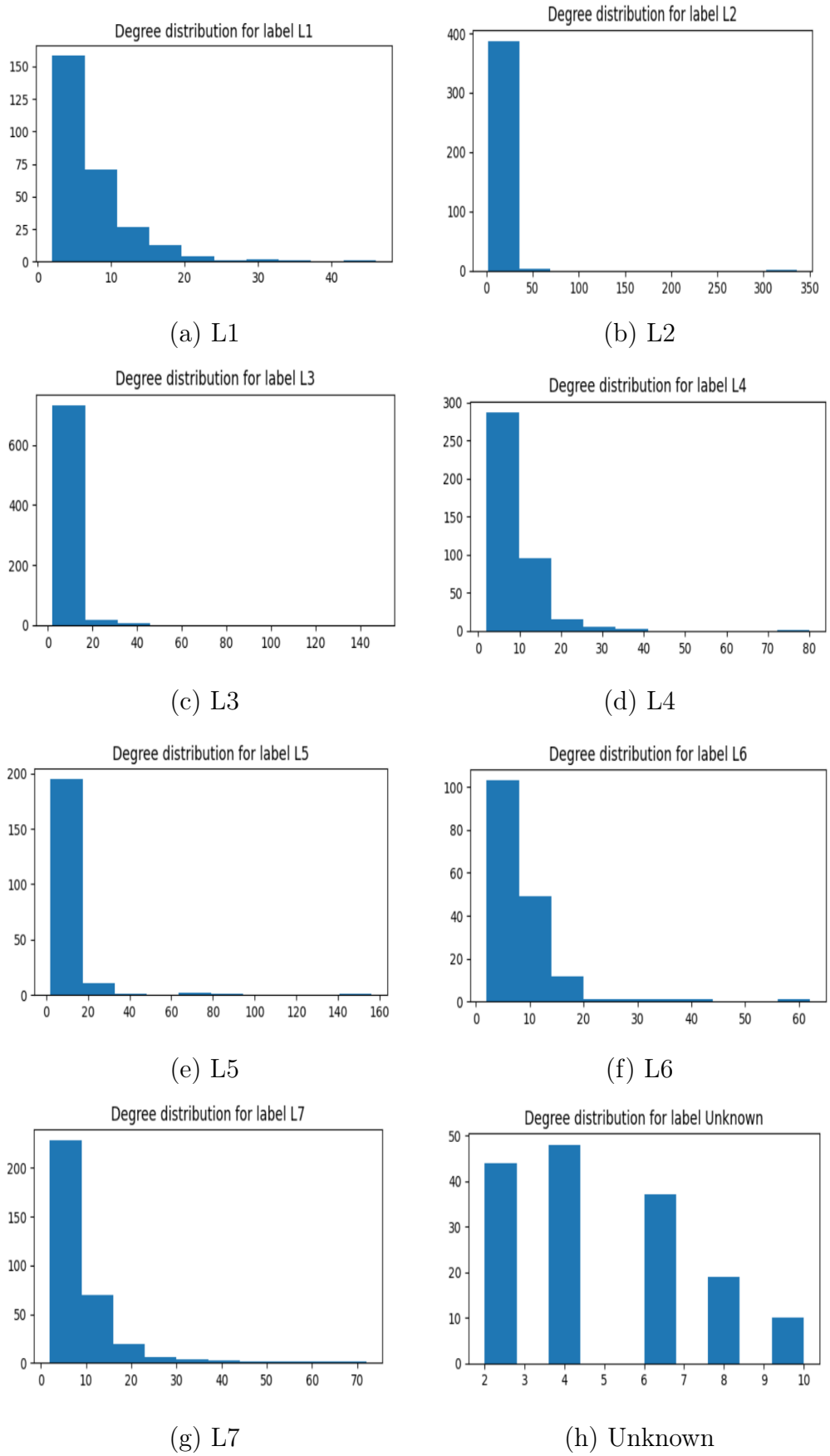
TABLE 2.6: Shortest Path Lengths for Each Label.

Label	**Mean**	**Median**	**Min**	**Max**
L6	0.1618792289033575	4	1	10
L1	5.574569924261621	6	1	12
L4	6.523063985933797	6	1	16
L5	3.8625574806075527	4	1	10
Unknown	6.627535946425054	6	2	17
L3	5.947301542789289 5	6	1	18
L7	5.292562414931106	5	1	15
L2	3.6556645476360883	4	1	10
Different	6.466525151685166	6	1	19

The label values with the lowest average shortest path lengths are L2 and L5. This suggests that nodes with these label values tend to be closer together in the graph than nodes with other label values. The label values with the highest average shortest path lengths are Unknown and L4. This suggests that nodes with these label values tend to be further apart in the graph than nodes with other label values.

The results also show that the average shortest path length for pairs of nodes with different label values is higher than for pairs of nodes with the same label value. This suggests that nodes with different label values tend to be further apart in the graph than nodes with the same label value.

C) I have also found the degree distribution chart for nodes with the same label. You can see their charts in the below images 2.5.



(a) L1 (b) L2

(c) L3 (d) L4

(e) L5 (f) L6

(g) L7 (h) Unknown

FIGURE 2.5: Degree Distribution Charts for Labels.

D) I have used several methods for finding the communities in the network, such as **Girvan-Newman**, **Label Propagation**, and **Greedy Modularity**. The results which have been calculated show the proportion of nodes with each label value in each community detected by different algorithms in our network. These proportions can help us understand if there is any correspondence between the communities and the label values of the nodes.

From the results of **Girvan-Newman**, it appears that some communities are composed mostly of nodes with a single label value, while other communities contain nodes with multiple label values. For example, Community 1 is composed mostly of nodes with label L6 (96%), while Community 0 contains nodes with multiple label values.

If most of the communities detected by a community detection algorithm contain nodes with only one label value, it suggests that there is a strong correspondence between the label values of the nodes and the community structure of the network. In this case, we could say that the label values of the nodes provide a good classification of the nodes into communities.

However, it's important to note that community detection algorithms are not perfect and may not always detect the "true" communities in a network. The results of community detection can vary depending on the algorithm used and the structure of the network. It's possible that a different algorithm or a different level of the hierarchy would give us different results.

2.5 Fifth Step

For this part I have use **Monte Carlo** simulation. I have also consider two scenarios. In the first one, I set the epidemic model to be in **SI** format, and the second one, I checked the simulation for **SIR**.

A **Monte Carlo** simulation is a method for estimating the probability of an event by running a large number of simulations and counting the number of times the event occurs. In this case, we are using a Monte Carlo simulation to estimate the probability of infection for each node with an Unknown label by running multiple simulations of the spread of infection on the graph and counting the number of times each node becomes infected. There are also two parameters in this algorithm, `beta`, and `num_simulations`. They have a direct relation with probability of infection. In the **SI** model, many of the nodes with **Unknown** label would have the $P(to\_be\_infected) = 1$, but In **SIR** model, the probability to become infected is between 0 and 0.01 for most of the nodes.

2.6 Sixth Step

For this part, I have tried these three algorithms to predict the labels of nodes which have **Unknown** label.

- Decision Tree
- Other Nodes in Community
- Label Propagation Based on Neighbors

A) The code which I have used for label prediction using decision tree, uses a decision tree classifier from the `scikit-learn` library to predict the labels of nodes with an Unknown label in a graph. A decision tree is a type of machine learning model that can be used for classification or regression tasks. It works by recursively splitting the data into subsets based on the values of the input features, and then assigning a label to each subset based on the most common label in that subset.

In this code, the decision tree classifier is trained on a set of labeled nodes in the graph. The input features for each node are its **degree**, **clustering coefficient**, **betweenness centrality**, and **closeness centrality**. These features are calculated using functions from the `networkx` library. The label for each node is its known label (L1 to L7).

Once the decision tree classifier has been trained on the labeled nodes, it can be used to predict the labels of the nodes with an **Unknown** label. The code does this by extracting the features for each **Unknown** node and passing them to the predict method of the classifier. The predicted label is then assigned to the node.

The `scikit-learn` library provides an implementation of decision trees that can be used for classification or regression tasks. The `DecisionTreeClassifier` class is used for classification tasks, and it provides several methods for training and using decision tree classifiers.

B) In the second approach, I have tried to find the most common label using the labels of other nodes in the community which is the current node is located.

C) Another strategy I have used to label **Unknown** nodes in the network is to use a **Label Propagation** algorithm. This approach involves iteratively updating the labels of **Unknown** nodes based on the labels of their neighbors until the labels converge or a maximum number of iterations is reached. The algorithm runs for a maximum of `max_iterations` iterations or until the labels converge.