PJ1

December 30, 2024

1 Network Analysis of Zachary's Karate Club

Network analysis has various applications, including social structure analysis, fraud detection, and more. In this notebook, you will analyze the structure of graphs using the `NetworkX` package. You will use the popular Zachary's karate club dataset (source).

Wayne W. Zachary [1] analyzed the social network of a university karate club from 1970 to 1972. Documenting interactions among 34 members through 78 pairwise links, he found that during the study, a conflict arose between administrator "John A." and instructor "Mr. Hi" that eventually split the club into two factions. Half of the members formed a new club around Mr. Hi, and members from the other part found a new instructor or gave up karate. Using a capacitated network model, he accurately assigned almost all members to their respective groups after the split, demonstrating how network bottlenecks in information flow can drive organizational fission.

[1] W. Zachary, An information flow model for conflict and fission in small groups, Journal of Anthropological Research 33, 452-473 (1977)

In the first step of this template, you will load the data in a matrix format into a `NetworkX` graph. Afterward, you will explore the network and derive insights from it in steps 2 and 3. Finally, you will visualize these insights in step 4.

[2]:
```
# Load packages
import matplotlib.pyplot as plt
import numpy as np
import networkx as nx
import pandas as pd
import seaborn as sns
color = sns.color_palette()
```

1.1 1. Load your data

1.1.1 Load data from file

The data should be in matrix format. E.g:

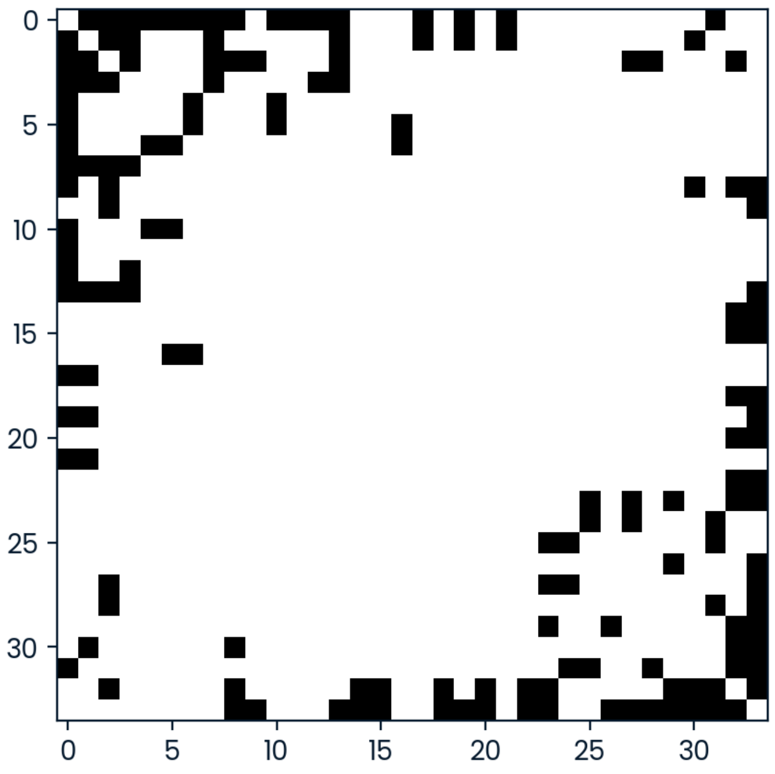
$$\begin{matrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{matrix}$$

where every element is seperated by a comma in a csv file.

This network is symmetric and would indicate that person 1 and 2 are connected, so are person 1 and 5, and so on.

We will load the very well-known Zachary's karateclub dataset. This dataset is also symmetric, which means that if person 1 is connected to person 2, person 2 is also connected to person 1.

[3]:
```
# Load data into numpy matrix
FILENAME = "zachary_karateclub.csv"
data=np.loadtxt(open(FILENAME, "rb"), delimiter=",")

# Show matrix to check if it is correctly loaded
# A black dot indicates that there is a connection between the two persons
_=plt.imshow(data, cmap='gray_r', aspect='equal')
```



1.1.2 Load data into networkx graph

```
[4]: # Set up networkx graph G from numpy matrix
     G = nx.from_numpy_matrix(data)
```

1.2 2. Explore the network

Now that the data has been loaded in a graph correctly, we can start exploring it.

Number of edges and nodes in graph

```
[5]: print(f'The number of edges in the graph: {len(G.nodes())}')
     print(f'The number of nodes in the graph: {len(G.edges())}')
```

```
The number of edges in the graph: 34
The number of nodes in the graph: 78
```

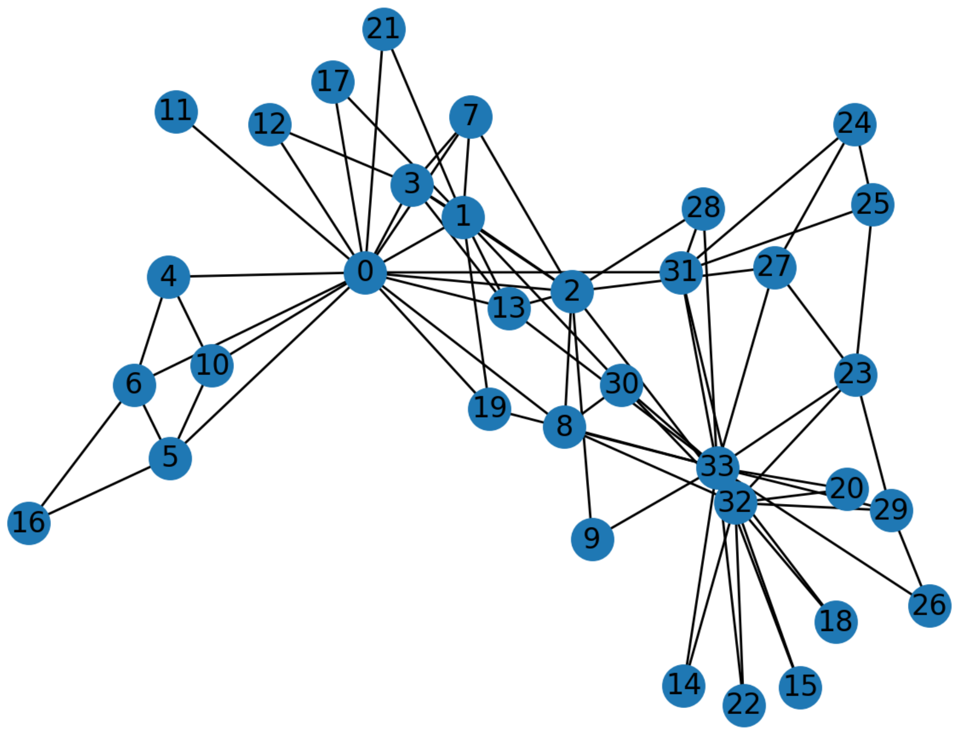
List of neighbors for any node

```
[6]: # Node to determine the neighbors for
     node = 2
     print(f'The neighbors of node {node} are {list(G.neighbors(node))}')

     # You can also print it for every node
     #for node in G.nodes():
     #    print(f'The neighbors of node {node} are {list(G.neighbors(node))}')
```

```
The neighbors of node 2 are [0, 1, 3, 7, 8, 9, 13, 27, 28, 32]
```

Show a basic visualization of the graph

```
[7]: # Show basic visualization of graph
     nx.draw(G, with_labels=True)
```



> **Note:** Network of the Zachary Karate Club. Distribution by degree of the node. Node 0 stands for the instructor "Mr. Hi", node 33 for the administrator "John A."

1.3 3. Derive insights from the graph

1.3.1 3.1 Node Connectivity

Which nodes in the network have the highest degree of connectivity?

Goal: Highlight important or influential nodes within the network and identify possible central figures.

Degree centrality (DC) is a measure of local centrality that is calculated from the immediate neighborhood links of a vertex. Recall that a node's degree is simply a count of how many social connections (i.e., edges) it has. The idea is to compute the normalized number of links incident to a node:

$$DC(v_i) = \frac{1}{N-1}\sum_{j=1}^{N}\alpha_{ij}$$

where v_i is the i-th node, N the number of nodes, and α_{ij} the number of edges between v_i and v_j. In a simple graph, DC value is between 0 and 1.

> **Note**: The degree centrality values are normalized by dividing by the maximum possible degree in a simple graph $N-1$, where N is the number of nodes in the graph. For

multigraphs or graphs with self-loops, the maximum degree might be higher than $N-1$, and values of degree centrality greater than 1 are possible.

[48]:
```
# Calculate degree centrality
node = 0

deg_cent = nx.degree_centrality(G)
print(f'The degree centrality for node {node} is {deg_cent[node]:.2f}')

# You can also print it for every node
#for node in G.nodes():
#    print(f'The degree centrality for node {node} is {deg_cent[node]:.2f}')
```

```
The degree centrality for node 0 is 0.48
```

For degree centrality, **higher values** mean that the node is more central. As mentioned above, each centrality measure indicates a different type of importance. Degree centrality shows how many connections a person has. They may be connected to many people at the heart of the network, but they might also be far off on the network's edge.

For example, in the Figure below, both nodes labeled "Bob" have the same high degree (i.e., lots of social connections, 9 in this case), but their roles are very different. The one in (b) is very central, and the one in (a) is peripheral. These show that while degree centrality accurately tells us who has a lot of social connections, it does not necessarily show who is in the "middle" of the network.

The **degree centrality** is an indicator for local opinion leaders, the possible positive application is restricted to small clusters of users like a forum. The obvious deduction about a user with high DC is a frequent communicator directly with other users that have the possibility of influencing them (Bodendorf & Kaiser, 2010). Therefore, DC is an acceptable measure of the immediate rate of influence spread from nodes in a short-term perspective. To detect influential nodes more accurately, we may formulate new measurements, for example, ClusterRank (CR) proposed by Chen, Gao, Lü, & Zhou (2013).

1.3.2 3.2 Network Communities

Communities in the network likely represent groups with closer or more frequent interactions. For example, in a social network, these might be friend groups; in an organizational network, departments, or project teams. Analyzing the connections between communities can reveal bridge nodes (nodes connecting communities), which are critical for maintaining information flow across the network.

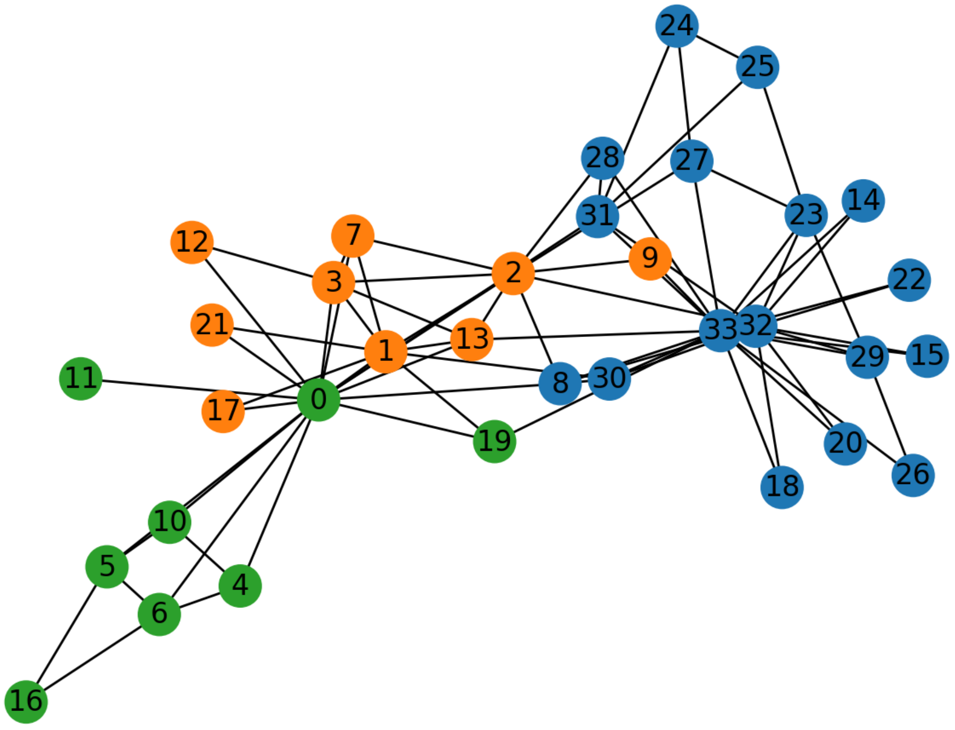
[9]:
```
# Import the community package from networkx
import networkx.algorithms.community as nxcom
```

[10]:
```
communities = sorted(nxcom.greedy_modularity_communities(G), key=len,␣
 ↪reverse=True)

for i, com in enumerate(communities):
    print(f'Community {i}: {list(com)}')
```

```
Community 0: [8, 14, 15, 18, 20, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33]
Community 1: [1, 2, 3, 7, 9, 12, 13, 17, 21]
Community 2: [0, 16, 19, 4, 5, 6, 10, 11]
```

Visualize communities within graph

[11]:
```
COLORS = ['tab:blue', 'tab:orange', 'tab:green', 'tab:red', 'tab:purple',
          'tab:brown', 'tab:pink', 'tab:gray', 'tab:olive', 'tab:cyan']

def get_index(node, node_groups):
    i = 0
    while(i<len(node_groups)):
        if node in node_groups[i]:
            return i
        else:
            i+=1
    return -1



communities = sorted(nxcom.greedy_modularity_communities(G), key=len,␣
 ↪reverse=True)

color_map = []

for node in G:
    index = get_index(node, communities)
    if(index >= len(COLORS)):
        print("More groups than colors; add some more colors.")
        break
    if(index == -1):
        print("Node note in a community")
        break
    color_map.append(COLORS[index])

nx.draw(G, node_color=color_map, with_labels=True)
plt.show()
```



1.3.3 3.3 Shortest Path Analysis

Identify efficient routes for information flow and how certain nodes serve as intermediaries

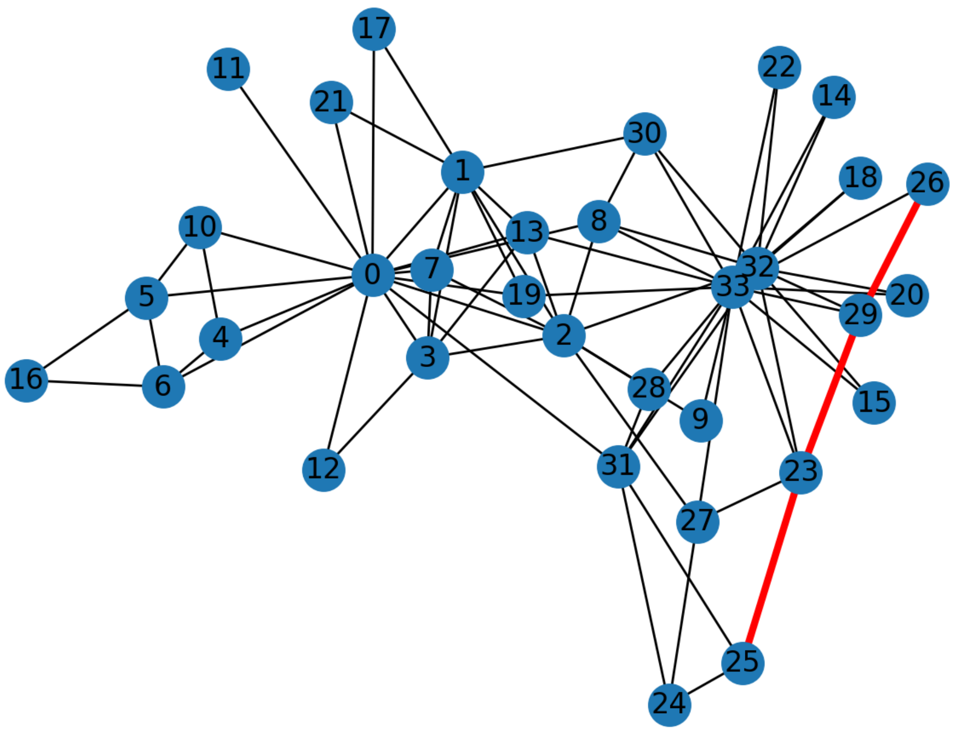
Shortest path between two specific nodes

[12]:
```
START_NODE = 29
END_NODE = 16

shortest_path = nx.shortest_path(G, START_NODE, END_NODE)

print(f'The shortest path from node {START_NODE} to node {END_NODE} is:␣
 ↪{shortest_path}')
```

```
The shortest path from node 29 to node 16 is: [29, 32, 2, 0, 5, 16]
```

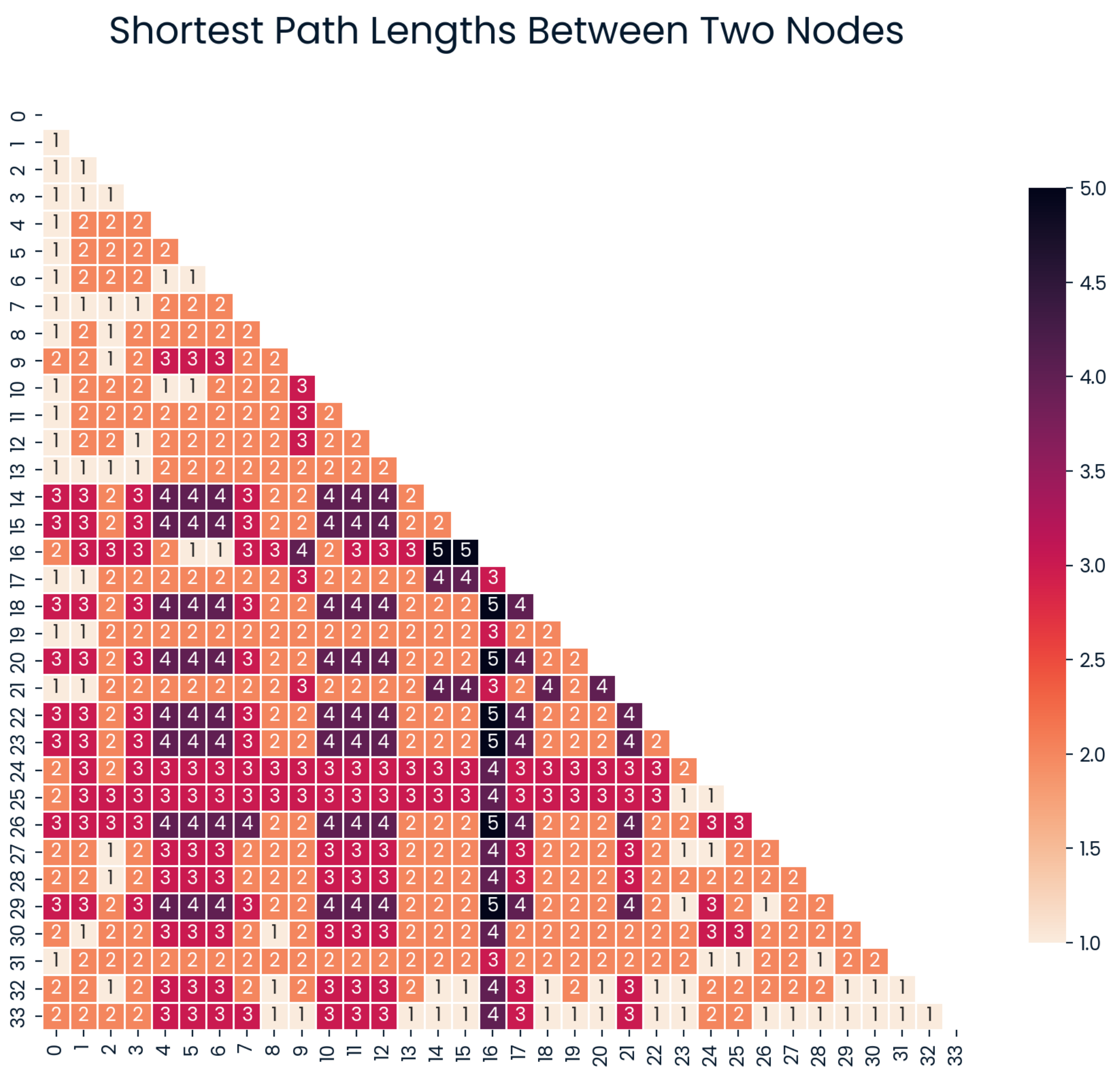
Highlight shortest path betwoon two specific nodes

[13]:
```
from itertools import combinations

def convert_nodes_to_edges(nodes_list):
    edges_list = []
    for i, n in enumerate(nodes_list):
        if(i<len(nodes_list)-1):
            next_n = nodes_list[i+1]
            edges_list.append((n, next_n))
    return edges_list

START_NODE = 26
END_NODE = 25

shortest_path = nx.shortest_path(G, START_NODE, END_NODE)

pos = nx.spring_layout(G, seed=7)
nx.draw(G, pos=pos, with_labels=True)

shortest_path_edge_list = convert_nodes_to_edges(shortest_path)

_=nx.draw_networkx_edges(G, pos, edgelist=shortest_path_edge_list, width=3,␣
 ↪edge_color='red')
```



Shortest path lengths between nodes

```
[14]: b = nx.shortest_path_length(G)
      #shortest_paths_df = pd.DataFrame()
      shortest_paths_dict = {}
      for spl in b:
          shortest_paths_dict[spl[0]] = spl[1]

      shortest_paths_df = pd.DataFrame.from_dict(shortest_paths_dict)
      shortest_paths_df.sort_index(inplace=True)

      # Generate a mask for the upper triangle
      mask = np.triu(np.ones_like(shortest_paths_df, dtype=bool))

      # Set up the matplotlib figure
      fig, ax = plt.subplots(figsize=(11, 9))                       # Set figure size

      # Generate a custom diverging colormap
      cmap=sns.color_palette("rocket_r", as_cmap=True)

      sns.heatmap(shortest_paths_df,
                  mask = mask,
                  cmap = cmap,
                  square = True,                                    # Ensure perfect␣
        ↪squares
                  linewidths = 0.5,                                 # Set linewidth␣
        ↪between squares
                  cbar_kws = {"shrink": .8},                        # Set size of color␣
        ↪bar
                  annot = True                                      # Include values␣
        ↪within squares
                 );

      plt.xticks(rotation=90)                                       # Rotate x labels
      plt.title('Shortest Path Lengths Between Two Nodes', size=20, y=1.05);   # Set␣
        ↪plot title and position
```



1.4 4. Exercises

Exercise 1. Write a Python function to calculate the degree centrality for each node, sort the nodes based on their degree centrality, and print the top 5 nodes. Who has the highest connectivity?

Exercise 2. Discuss the role of each community by answering the questions below.

```
(a) How tightly are they connected internally, and how do they connect to other communities?
(b) Are there any "bridge" nodes between communities, and what roles do they play?
(c) How could removing specific nodes impact inter-community connectivity? Write a Python func
```

Exercise 3. Write a python function for calculating the average shortest path length for the entire network. How does this value change if a high-centrality node is removed?

Exercise 4. Write a python function for finding the longest shortest path in the network, also known as the network diameter. Which nodes are involved, and what does this path tell you about network spread?

Exercise 5. Find the nodes that appear most frequently in shortest paths between other pairs of nodes. What roles do these nodes play in network connectivity?

Exercise 6. If a node is removed from the network, recalculate the shortest paths for affected node pairs. How does the removal impact the overall average shortest path length and network connectivity?

1.4.1 Acknowledgement

There is a lot more you can do with networks. Visit the NetworkX documentation for inspiration.