Graph Partitioning

Sofia Huang

due 11/15/2022

1. Color nodes based on final split

Draw the original Karate club graph (before the split) and color the nodes according to the factions they belong to (John A or Mr. Hi). This should look similar to the graph on slide 92 - all edges should be present, just indicate the nodes in the eventual split by color.

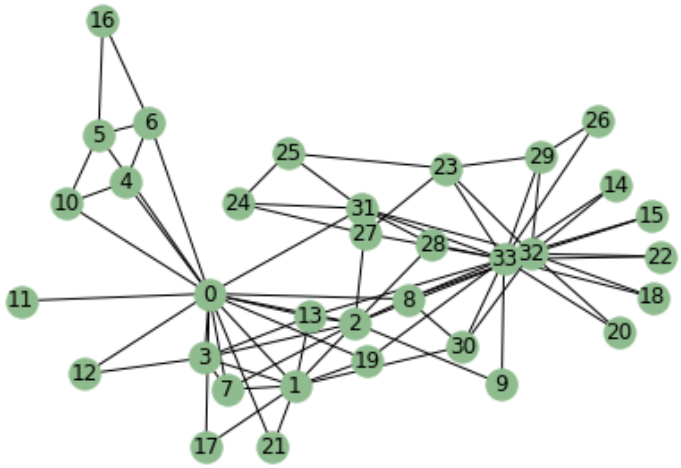


Figure 1: Original Karate Club graph.

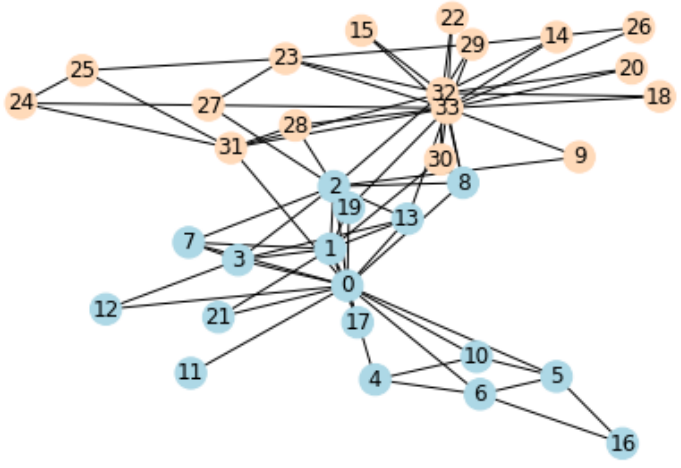


Figure 2: Split Karate Club graph.

Q: How many nodes (students) eventually go with John and how many with Mr. Hi?

17 students eventually go with John and 17 go with Mr. Hi.

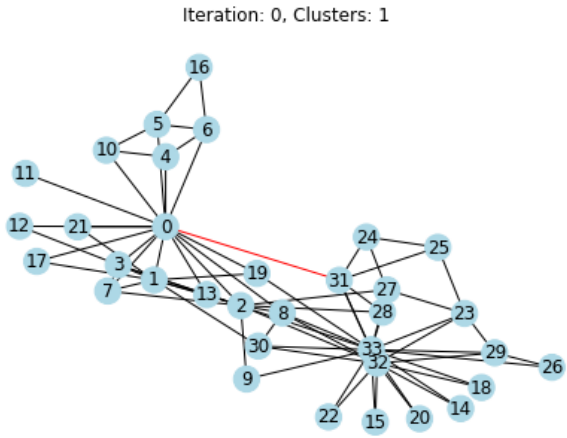
To draw the graphs, I used the python package Networkx and its built-in Karate Club graph function. Then, to color the nodes to represent the split, I created a dictionary that connected the club to the color I wanted those nodes to be. Then, I used list comprehension to create a list of colors that corresponds to the list of Karate Club graph nodes. Lastly, I drew the graph showing the split using the list of colors. The orange represents those who went with John and the blue represents the students who went with Mr Hi.

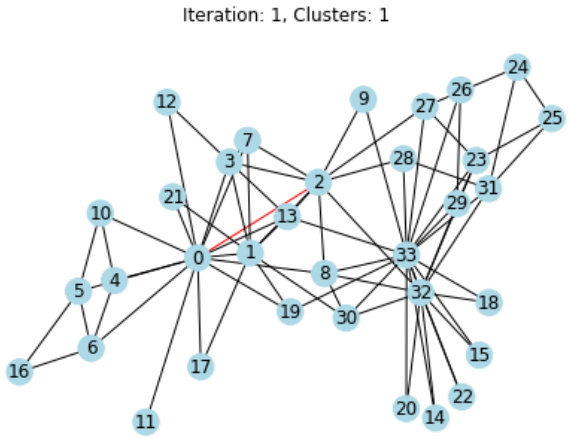
```
import networkx as nx
import matplotlib.pyplot as plt
K = nx.karate_club_graph()
club_color = {
    'Mr. Hi': 'lightblue',
    'Officer': 'peachpuff',
}
node_colors = [club_color[K.nodes[n]['club']] for n in K.nodes]
nx.draw(K, node_color=node_colors, with_labels=True)
```

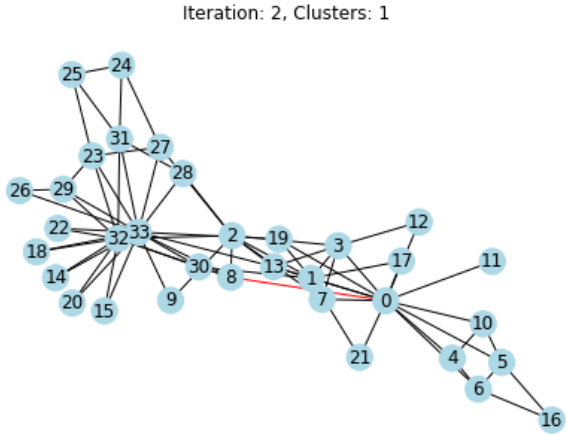
Listing 1: Drawing the split Karate Club graph

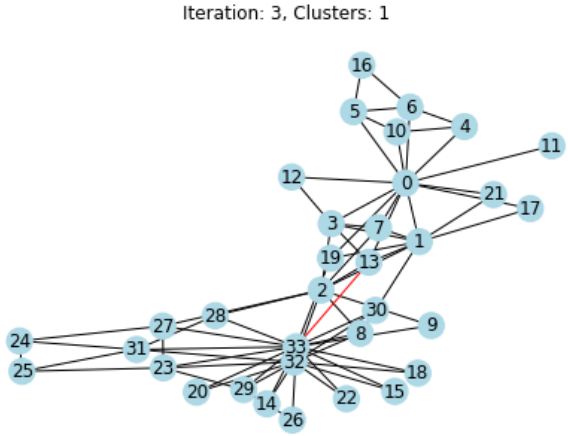
2. Use the Girvan-Newman algorithm to illustrate the split

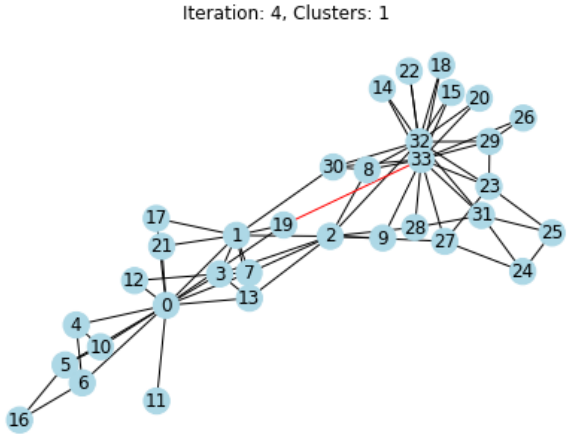
We know the final result of the Karate Club split, which you've colored in Q1. Use the Girvan-Newman algorithm to check if the split could have been predicted by the social interactions expressed by edges. How well does the mathematical model represent reality? Generously document your answer with all supporting equations, code, graphs, arguments, etc.
Keeping the node colors the same as they were in Q1, run multiple iterations of the Girvan-Newman graph partioning algorithm (see Module-07 Social Networks, slides 90-99) on the Karate Club graph until the graph splits into two connected components. Include an image of the graph after each iteration in your report.

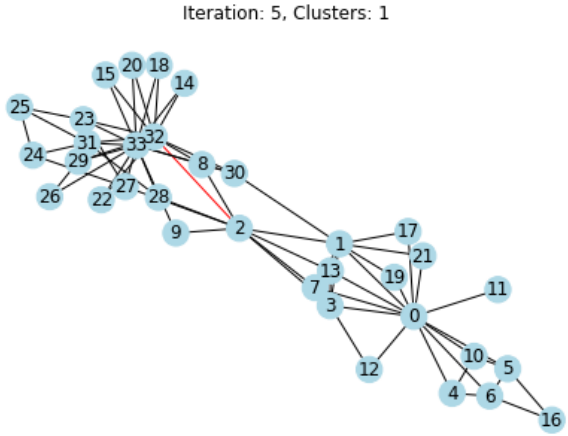

Iteration: 0, Clusters: 1

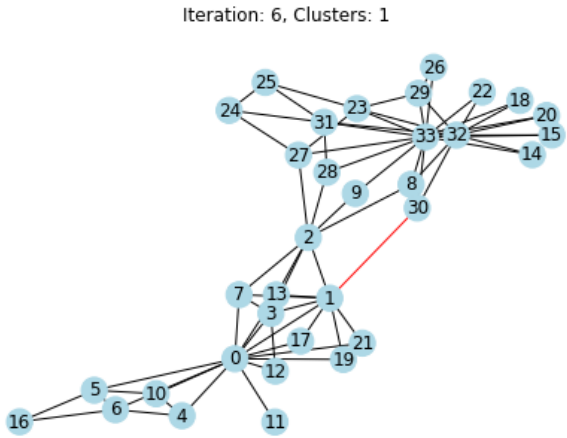

Iteration: 1, Clusters: 1

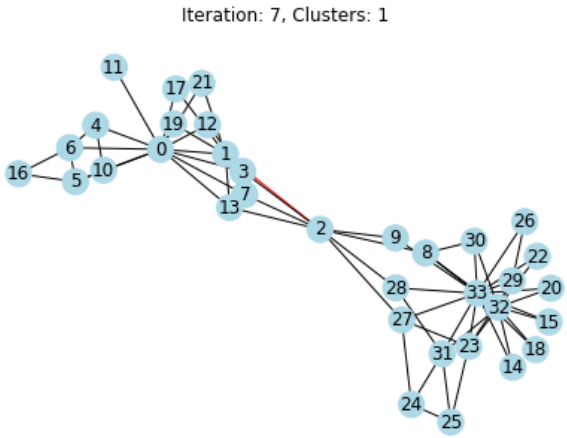

Iteration: 2, Clusters: 1

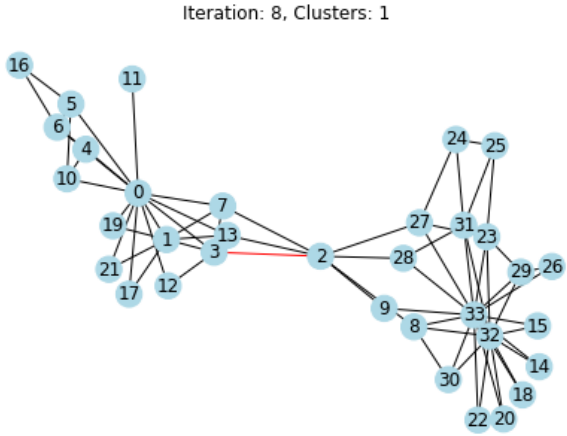

Iteration: 3, Clusters: 1

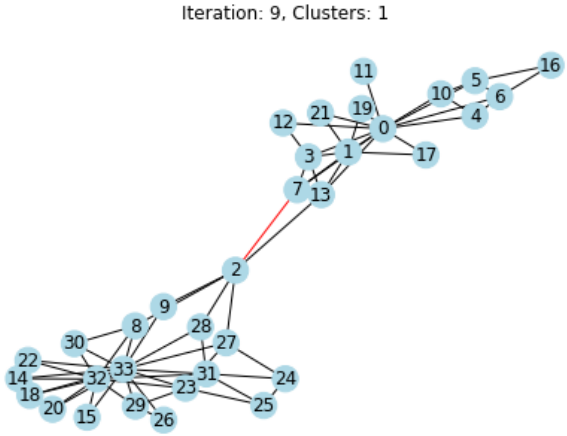

Iteration: 4, Clusters: 1

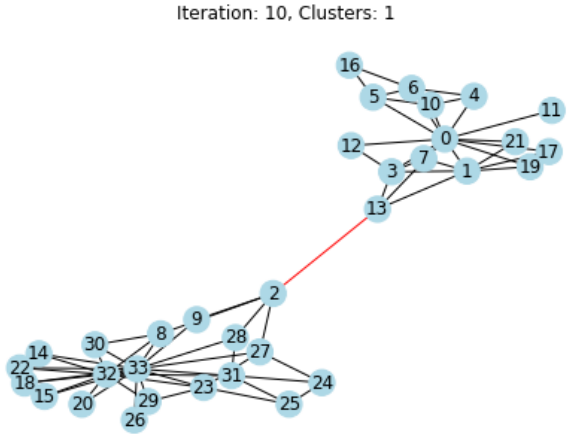

Iteration: 5, Clusters: 1

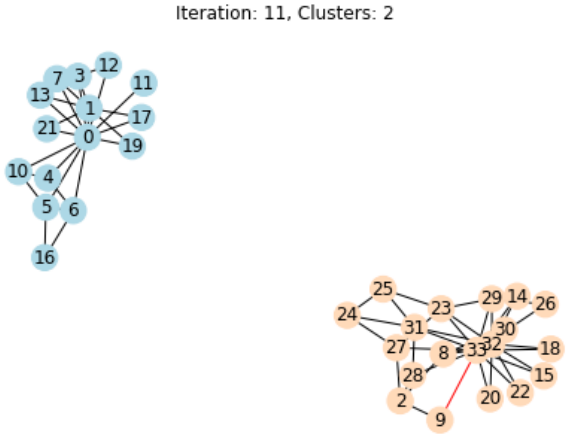

Iteration: 6, Clusters: 1


Iteration: 7, Clusters: 1


Iteration: 8, Clusters: 1


Iteration: 9, Clusters: 1


Iteration: 10, Clusters: 1


Iteration: 11, Clusters: 2

Q: How many iterations did it take to split the graph?

It took 11 iterations to split the graph into 2 partitions.

To replicate the Girvan-Newman algorithm, I calculated the betweenness value for all edges and found the edge with the highest value. Then, I drew the graph before removing the edge with the highest betweenness so I could change its color to show which edge was removed for each iteration. I repeated the steps until there were two clusters and the graph was partitioned. The final graph shows the result of the Girvan-Newman algorithm on the Karate Club graph, the 2 partitions showing who went with John vs Mr Hi. To accomplish this, I utilized the Networkx package to calculate edge betweenness and get the connected components.

```
def girvan_newman(graph):
  # find number of connected components
  partitions = nx.connected_components(graph)
  cluster_count = nx.number_connected_components(graph)
  count = 0

  while(cluster_count == 1):
    # calculate betweenness and find edge with the highest
    edge_betweenness = nx.edge_betweenness_centrality(graph)
    edge_to_remove = max(graph.edges(), key=edge_betweenness.get)

    # replace the color in edge_color_list with red if the edge has the
    maximum betweenness:
    edge_color_list = ["black"]*len(graph.edges)
    for i, edge in enumerate(graph.edges()):
        if edge == edge_to_remove or (edge[1],edge[0]) ==
   edge_to_remove:
            edge_color_list[i] = 'red'

    # find the nodes forming the connected components
    partition_nodes = []
    for node in partitions:
        partition_nodes.append(list(node))

    # create color map to show connected components
    color_map = []
    for node in graph:
        if node in partition_nodes[0]:
            color_map.append('lightblue')
        else:
            color_map.append('peachpuff')

    # plot the graph
    plt.figure(count)
    nx.draw(graph, node_color=color_map, edge_color = edge_color_list,
   with_labels=True)
```

```
34
35      # update the number of connected components
36      cluster_count = nx.number_connected_components(graph)
37      partitions = nx.connected_components(graph)
38
39      # save each graph
40      plt.tight_layout()
41      plt.title('Iteration: {}, Clusters: {}'.format(count, cluster_count
        ))
42      plt.savefig('/content/drive/MyDrive/Colab Notebooks/DATA 440 - Web
        Science/KC_GN{}.png'.format(count),bbox_inches='tight')
43
44      # remove the edge with the highest betweenness
45      graph.remove_edge(edge_to_remove[0], edge_to_remove[1])
46
47      count+=1
```

Listing 2: Girvan-Newman algorithm function

3. Compare the actual to the mathematical split

Compare the connected components of the Girvan-Newman split graph (Q2) with the connected components of the actual split Karate club graph (Q1).

Q: Did all of the same colored nodes end up in the same group? If not, what is different?

All of the same colored nodes except for 2 and 8 ended up in the same group. In the actual mathematical split, 2 and 8 were with Mr Hi. While, in the Girvan-Newman graph, 2 and 8 ended up with John.

References

- StackOverflow - Saving Matplotlib graph with title `https://stackoverflow.com/questions/64576843/why-when-saving-my-matplotlib-image-the-title-does-not-appear`
- StackOverflow - Coloring spcific edges on a graph `https://stackoverflow.com/questions/34120957/python-networkx-mark-edges-by-coloring-for-graph-drawing`
- Networkx - edge_betweenness_centrality `https://networkx.org/documentation/networkx-1.10/reference/generated/networkx.algorithms.centrality.`

edge_betweenness_centrality.html#networkx.algorithms.centrality.edge_betweenness_centrality

- Using the Graph data structure in Python https://www.section.io/engineering-education/graph-data-structure-python/#representing-graphs