Old Dominion University

Assignment Five

Joshua Gahan

March 24, 2019

1 Karate Club, comparision of algorithm performance with historical data.

We populated a graph with data from the Zachary's Karate Club study and then we applied Girvan-Newman's algorithm to split the group into two distinct communities. We then compared the results of our graph post split to the real data provided in the study to compare the accuracy of algorithm in predicting how communities might split up, given the relations between the members of the community.
We began by reading the known data in and populating our nodes. We chose to color the nodes based on the way that the community split up in the real study. Members of Mr Hi's community have had their nodes colored red, and members of John's community have had their nodes colored green. Mr. Hi and John's nodes where named HI and JA respectively.
We utilized the networkx library to manage our nodes and edges, and used matplotlib to draw our graphs.

1.1 Implementation of Girvan_Newman Algorithm.

```
for url in data:
    savefile_counter = 0


    all_members = set(range(34))
    club1 = {0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 13, 16, 17,
        19, 21}

    G = nx.Graph()
    G.add_nodes_from(all_members)
    G.name = "Zachary's Karate Club"

    # data read in here
    #
    #

    color_map = []
    for node in G:
        if node in club1:
            color_map.append('green')
        else:
            color_map.append('red')
```

```
labeldict = {}
edgedict = []
for i in range (0,len(G)):
    labeldict[i] = i

for i in range (0,len(G.edges)):
    edgedict.append('black')

print(G.edges)

labeldict[0] = "JA"
labeldict[33] = "HI"


nx.draw(G, node_color = color_map, edge_color=edgedict,
    width=2, labels=labeldict, with_labels=True)
plt.show()
plt.savefig('./graph/karate%s.png' % savefile_counter)


graphs = list(nx.connected_component_subgraphs(G))
while len(graphs) == 1:
    betweeness = nx.edge_betweenness(G, 34, normalized=True)
    sort = sorted(betweeness.items(), key=lambda x: x[1])

    leading_edge = sort[len(sort) - 1][0]

    edge_counter = 0
    for u, v in G.edges:
        if (u == leading_edge[0] and v == leading_edge[1]):
            edgedict[edge_counter] = 'blue'
            nx.draw(G, node_color=color_map,
                edge_color=edgedict, width=2,
                labels=labeldict, with_labels=True)
            plt.show()
            del edgedict[edge_counter]
            G.remove_edge(u, v)
            nx.draw(G, node_color=color_map,
                edge_color=edgedict, width=2,
                labels=labeldict, with_labels=True)
            plt.show()
            break
      else:
          pass
      edge_counter += 1
```

```
graphs = list(nx.connected_component_subgraphs(G))
```

We start off by creating 34 nodes which will later be populated with data. the numbers in club1 represent real-life individuals who stayed with John upon the clubs split. The nodes are id'd between 0 and 33, representing the 34 members of the original club. Thus, node 3 would be not only the third node, but would have '3' as its unique identifier.
We then create a color_map list to which we append either "red" or "green" depending on if the the respective node id is equal to one of the numbers in club1. This color list will be used during plotting to color our nodes appropriately. We performed similar operation to populate labeldict and edgedict which are the node labels and the edge colors respectively. Utilizing these helper lists, we then draw our graph.
We then come to the heart of the algorithm. Nx.connected.. returns an array containing each component in the graph, we take advantage of this by making the length of the connected array the condition of our while loop.
Within the loop, we utilize nx.edge_betweeness to calculate the betweeness of each of our edges. Once calculated, we sort the returned betweeness dictionary such that the item with the highest betweeness is the last item in the list. This dictionary has a tuple of nodes as its keys, and here we search for the edge that has both of the leading_edge tuple as its parents. We then color this edge blue for display and then remove the edge.
We create two additional plots. for each iteration through the while loop, representing the graph with the edge to be deleted colored blue, and then the edge removed in the next plot.

2 Findings

We have found that the Girvan-Newman algorithm does indeed do a very good job of predicting the split in the social graph. In the original study, Zachary managed to correctly predict how all but one of the members would split when the karate club broke up. The Girvan-Newman algorithm in our implementation successfully predicted how all but two members (two and eight) would break. Given the sheer number of variables that could affect a split in real life such as what this study followed, the algorithm performs amazingly well given that it simply operates by breaking edges depending on the degree of betweeness between nodes.

3 D3

A implementation showing the splitting of this graph along historical lines has been provided as one file labeled index.html. This implementation does not utilize the algorithm, but rather simply breaks according to the data provided. You will operate it by clicking the provided button multiple times, each time removing a single relevant edge.

4 The algorithm at work

What follows is a series of images, that shows the algorithm at work. Below a full showing of the algorithm splitting the karate club into 2 groups is example images which show the group ran through the same algorithm after being split into 3,4,5,6, and 10 nodes.

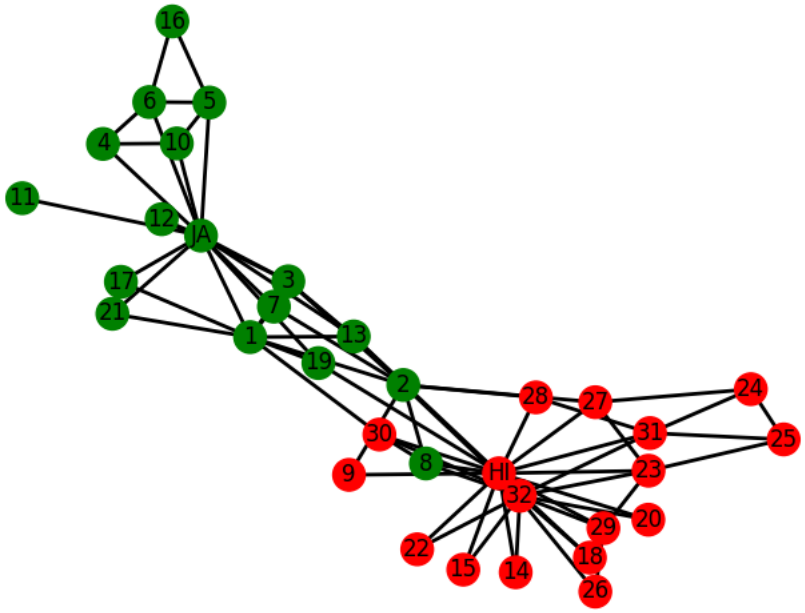

Figure 1: Graphical visualization of the algorithm at work

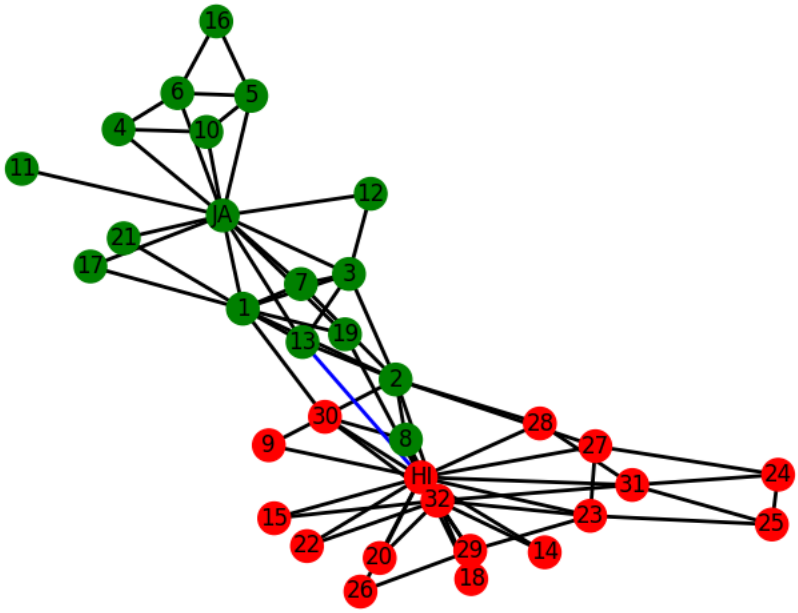

Figure 2: Graphical visualization of the algorithm at work

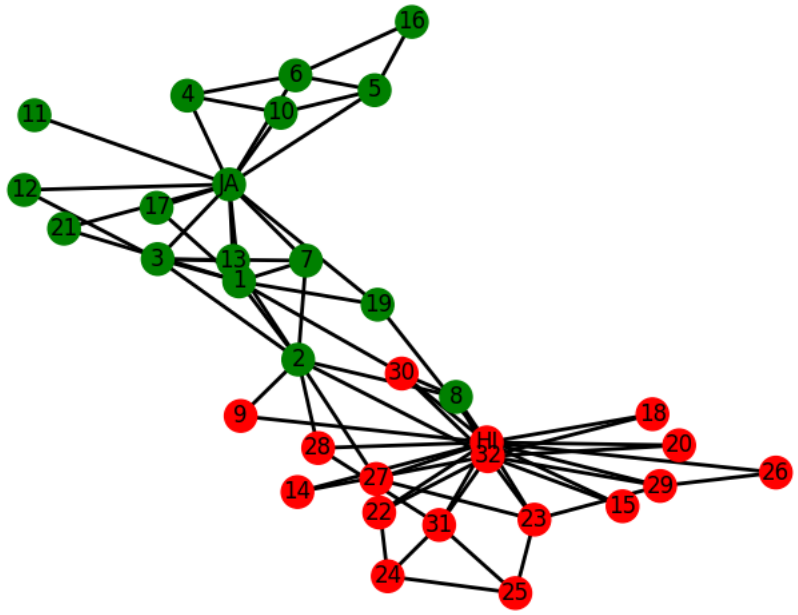

Figure 3: Graphical visualization of the algorithm at work

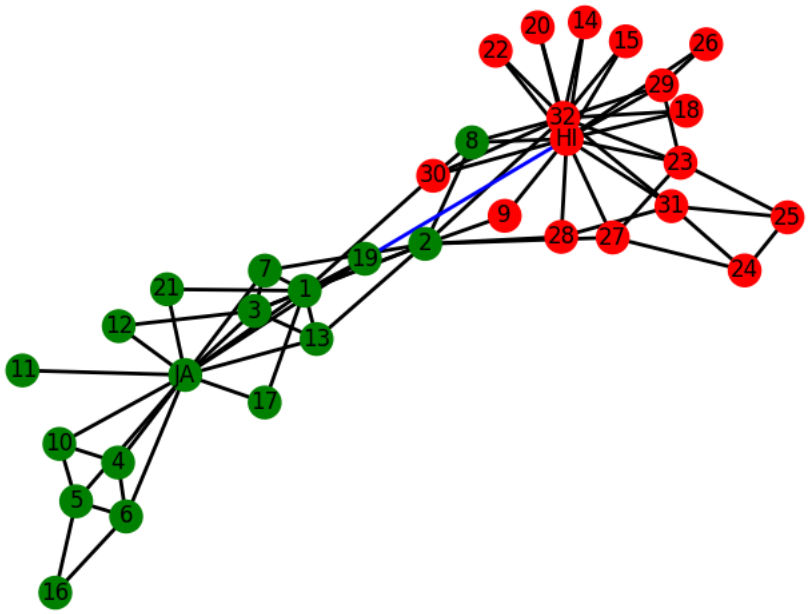

Figure 4: Graphical visualization of the algorithm at work

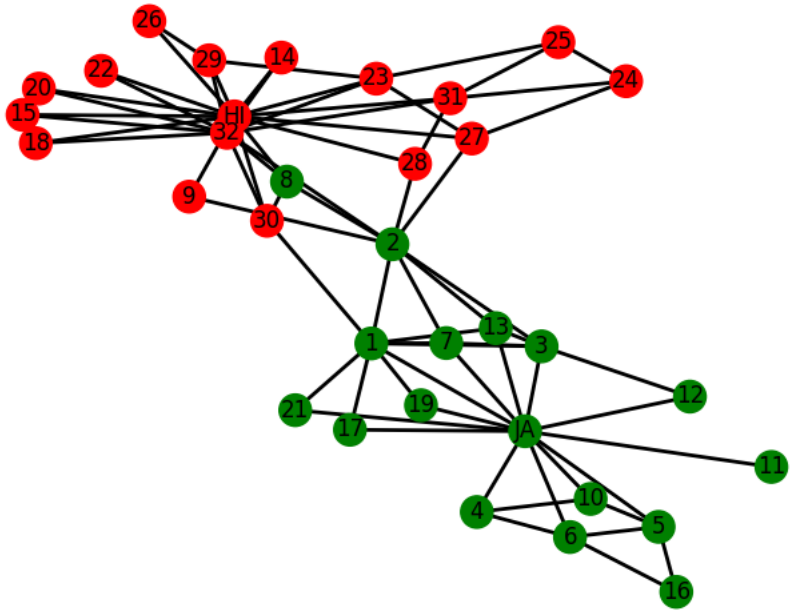

Figure 5: Graphical visualization of the algorithm at work

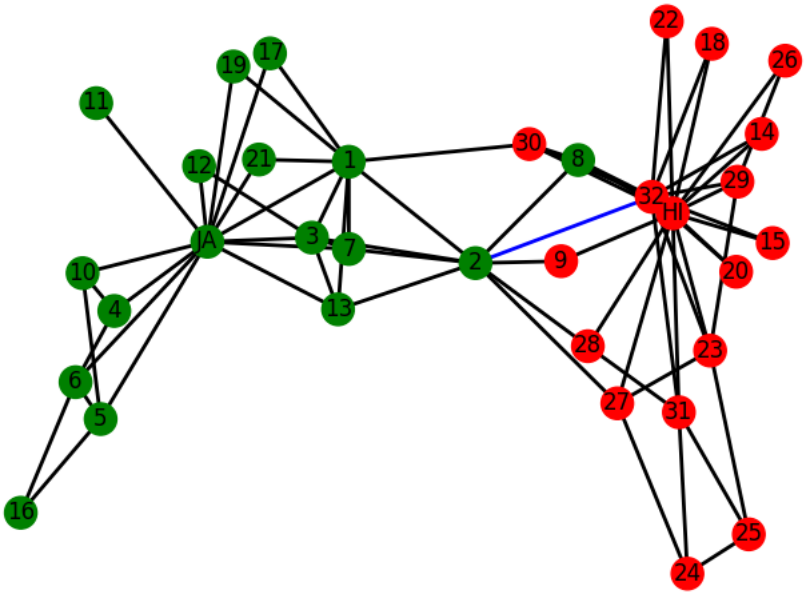

Figure 6: Graphical visualization of the algorithm at work

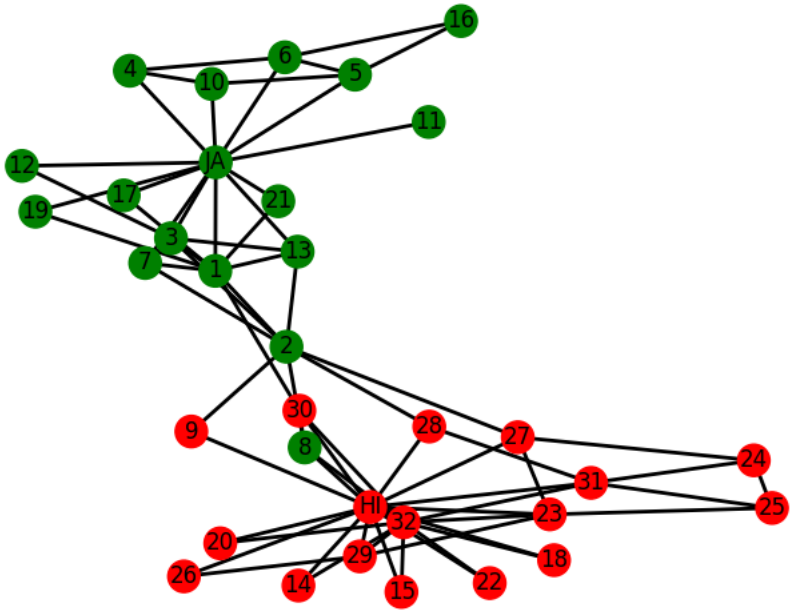

Figure 7: Graphical visualization of the algorithm at work

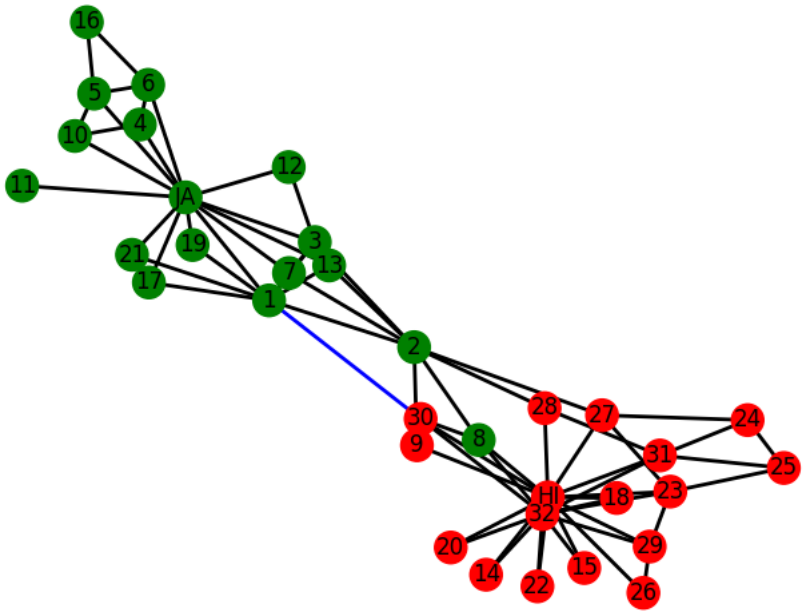

Figure 8: Graphical visualization of the algorithm at work

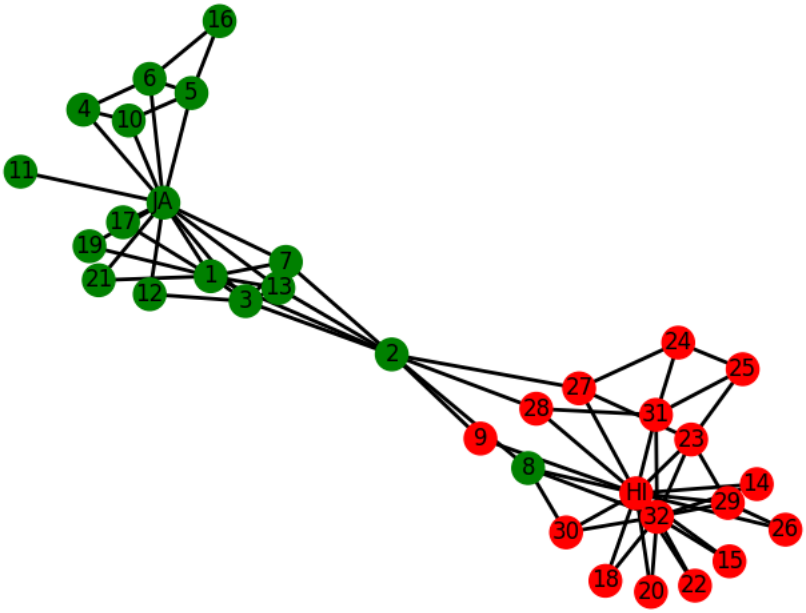

Figure 9: Graphical visualization of the algorithm at work

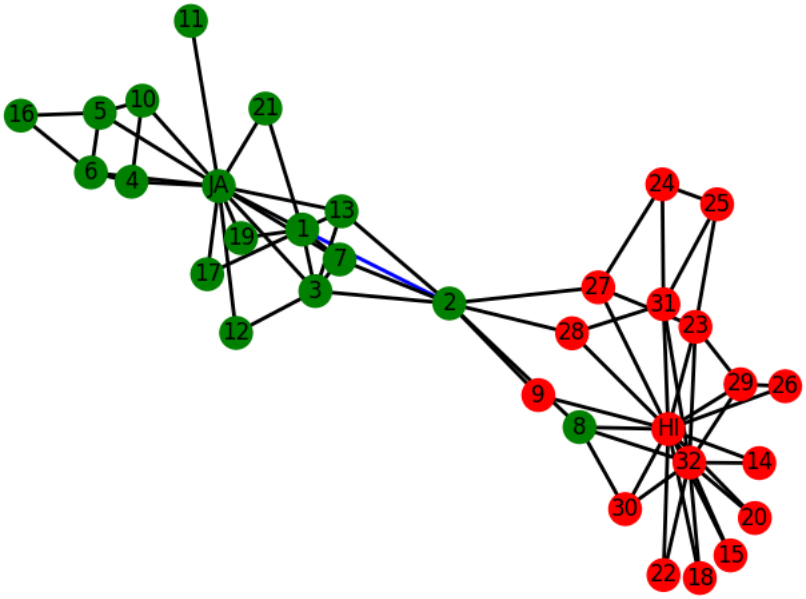

Figure 10: Graphical visualization of the algorithm at work

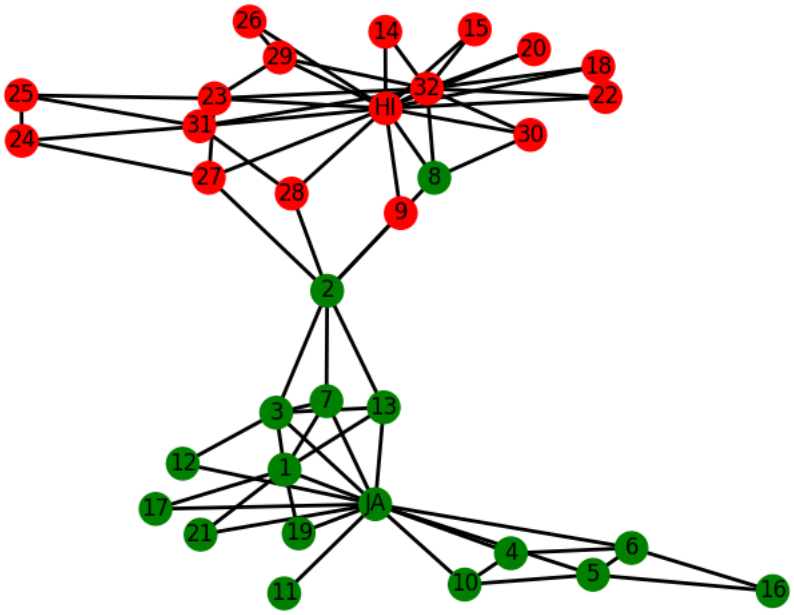

Figure 11: Graphical visualization of the algorithm at work

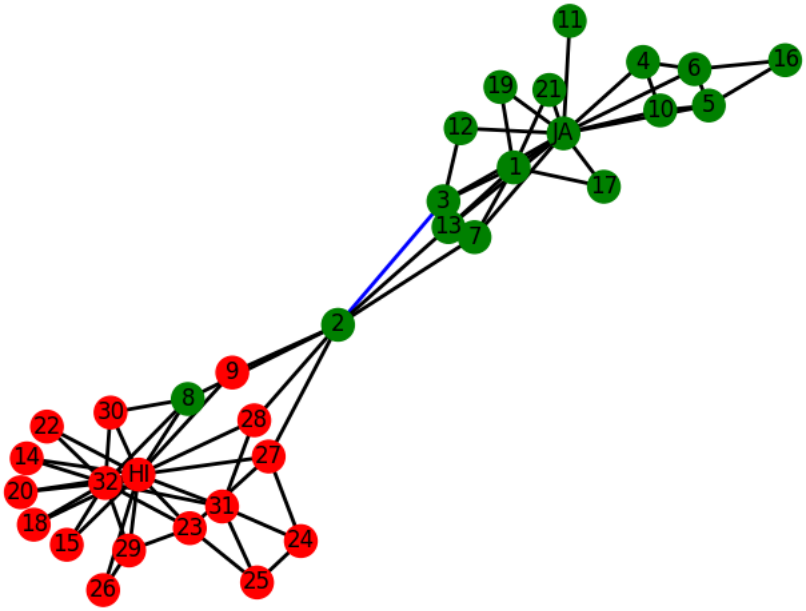

Figure 12: Graphical visualization of the algorithm at work

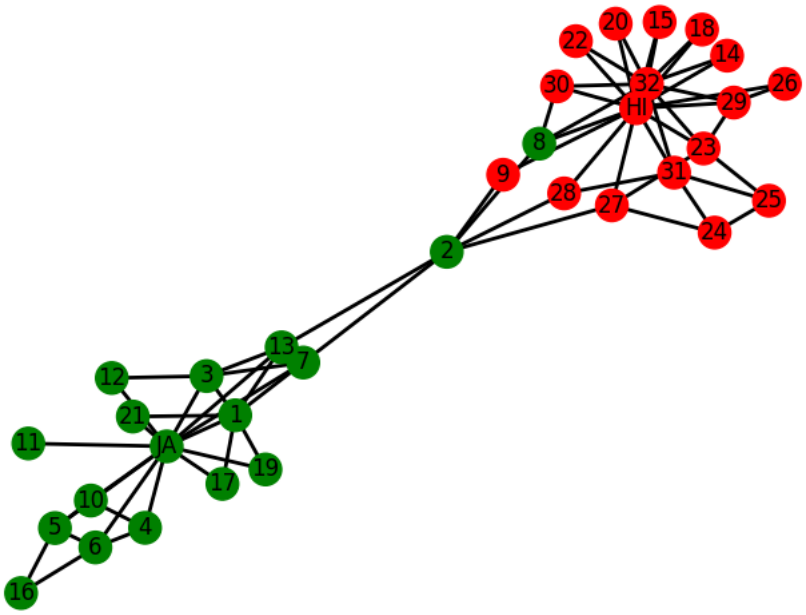

Figure 13: Graphical visualization of the algorithm at work

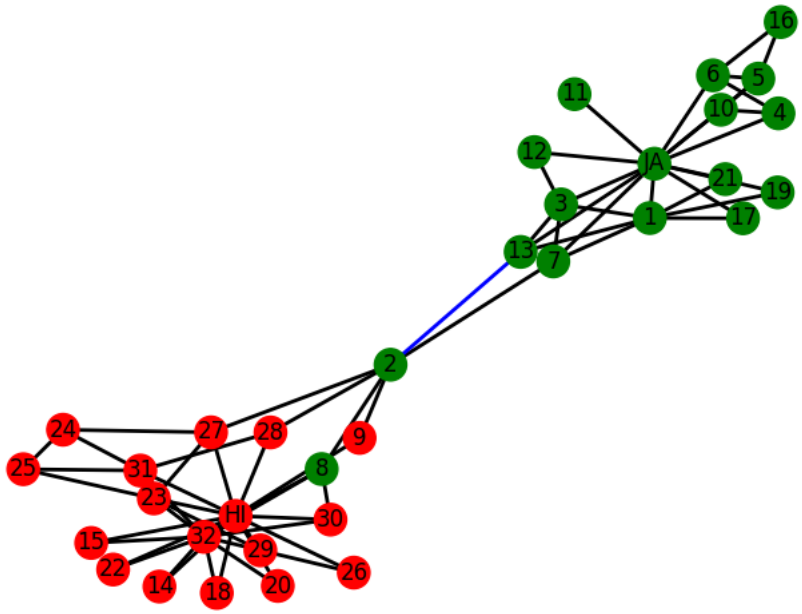

Figure 14: Graphical visualization of the algorithm at work

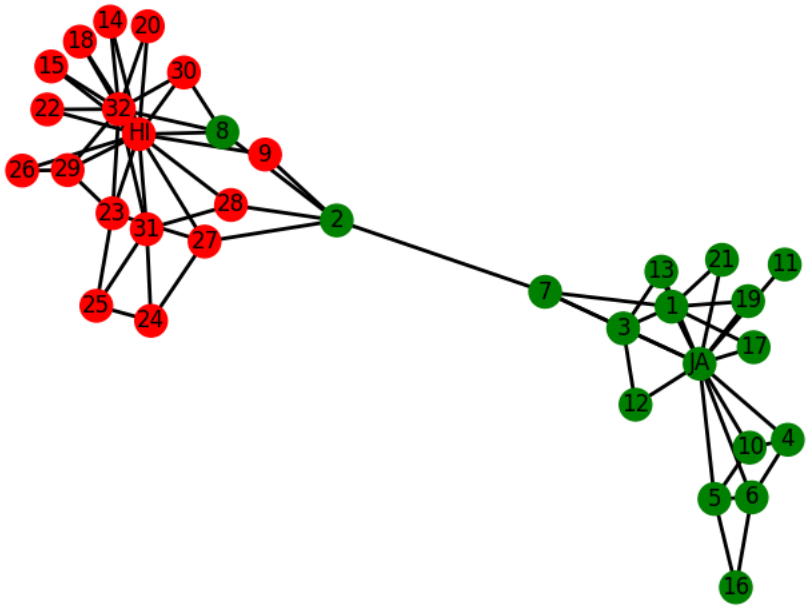

Figure 15: Graphical visualization of the algorithm at work

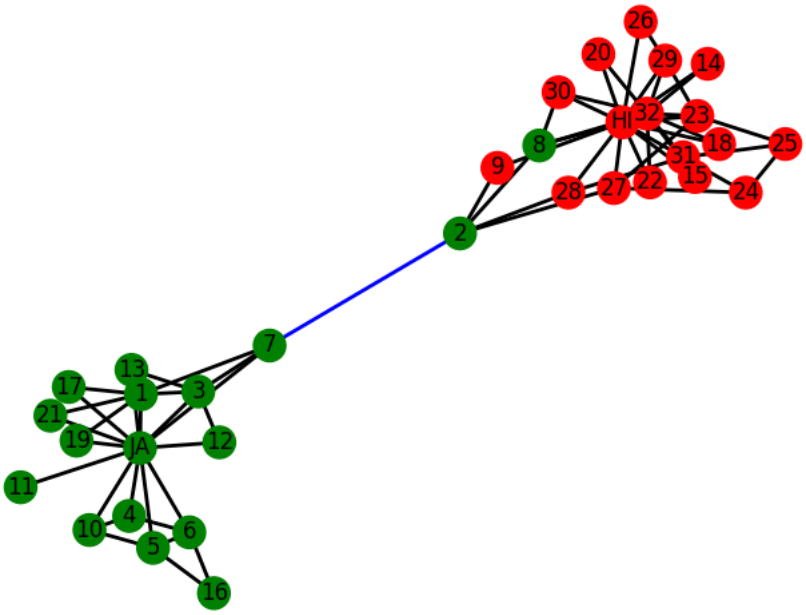

Figure 16: Graphical visualization of the algorithm at work

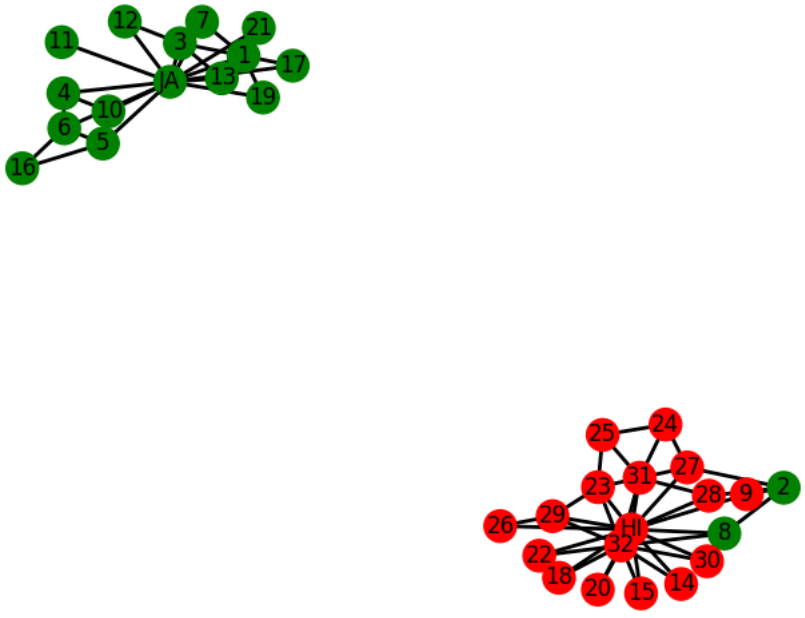

Figure 17: Graphical visualization of the algorithm at work

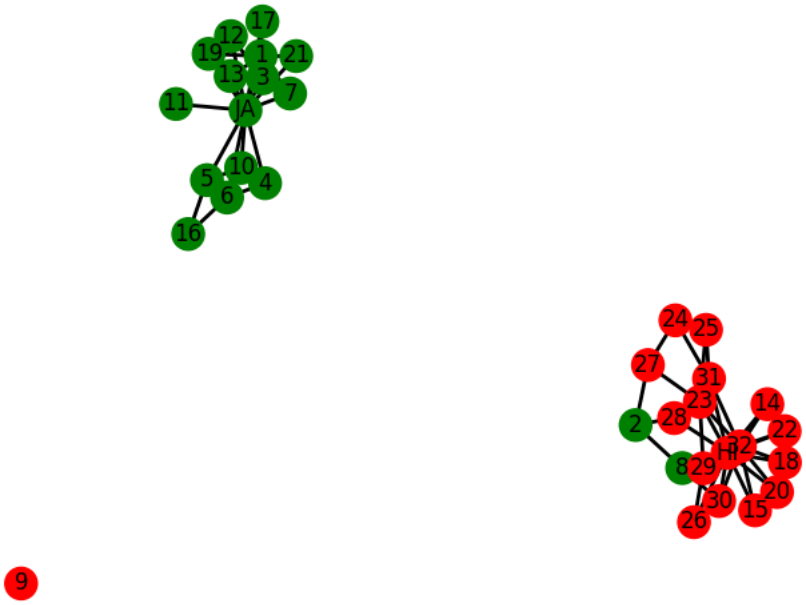

Figure 18: Split into 3 nodes

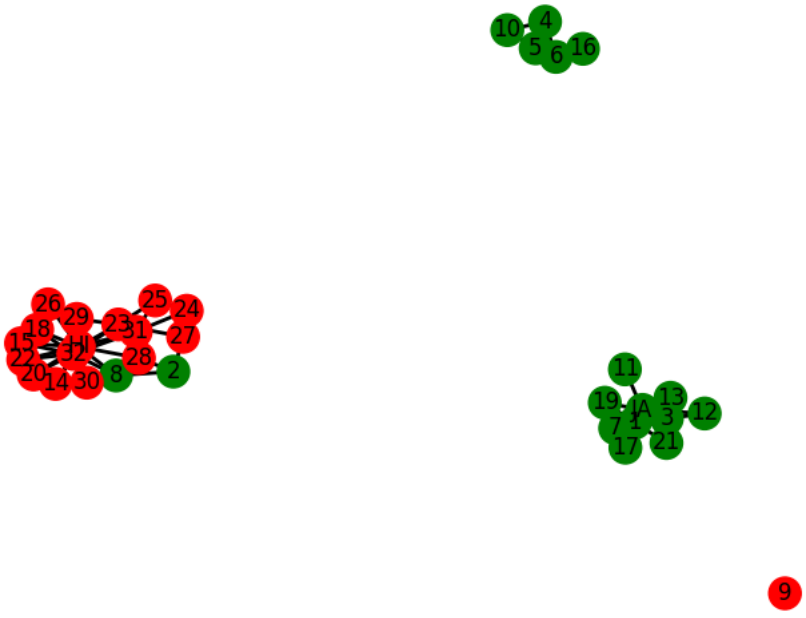

Figure 19: split into 4 nodes

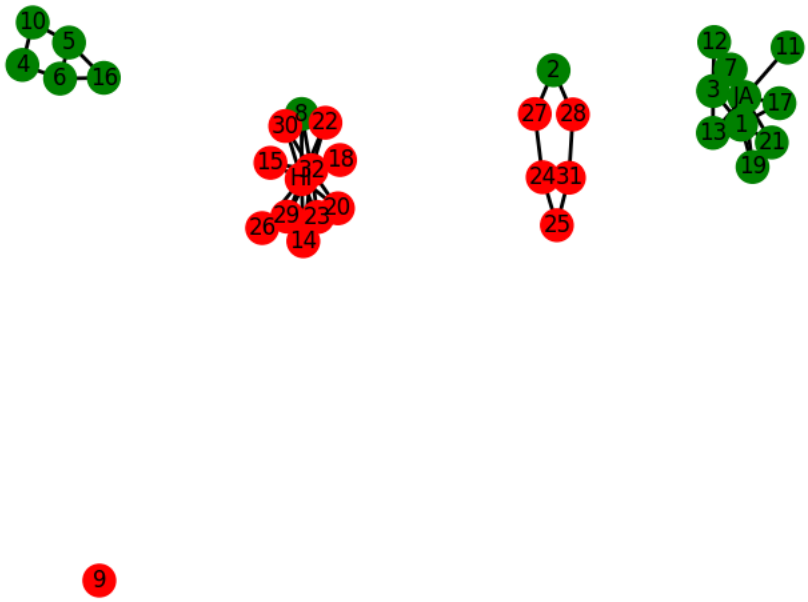

Figure 20: Split into 5 nodes

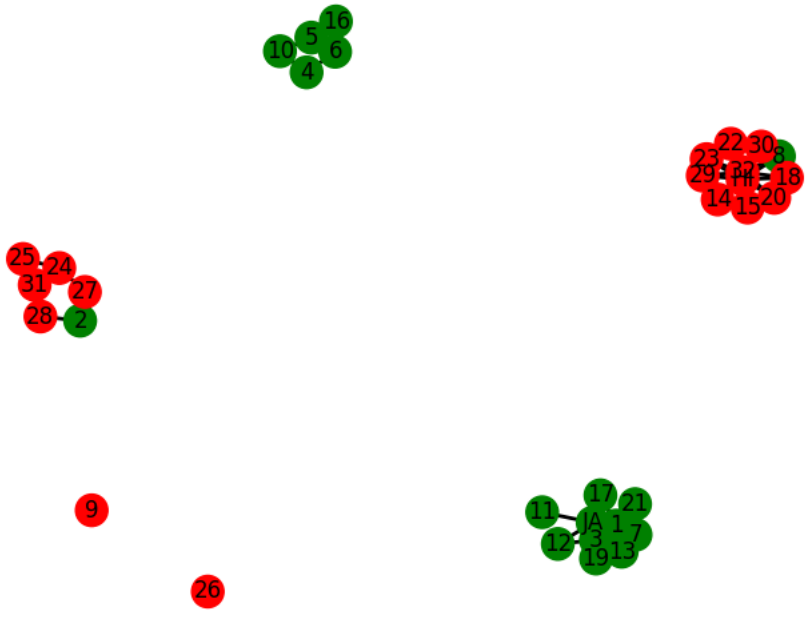

Figure 21: Split into 6 nodes

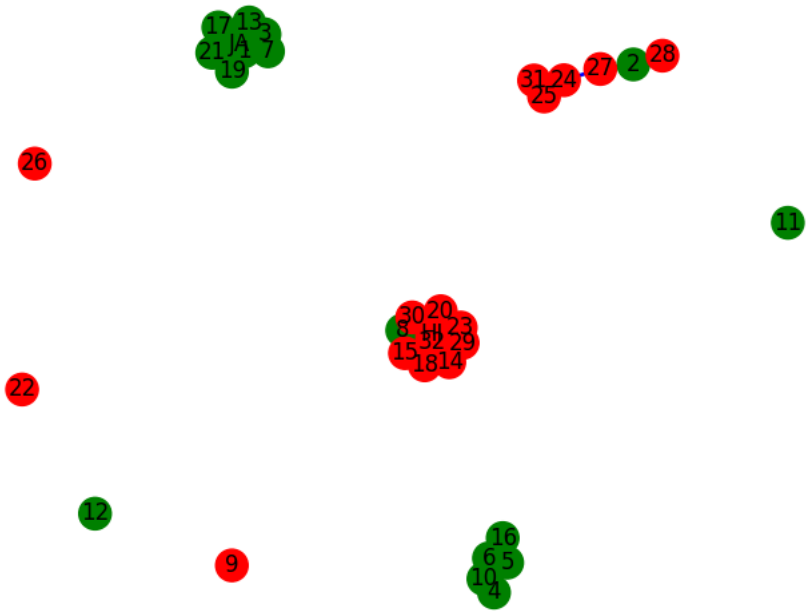

Figure 22: Split into 10 groups