CSCE 5013 - Graph Theory

010646947

Zack Fravel

1 Homophily analysis of U.S. Senate Votes

Below are the graph specifications we'll be using for analysis:

1. Each node corresponds to a Senator.
2. Edges are added if and only if Senators voted the same way.
3. The weight of each edge is the number of times each Senator pair voted similarly.

The following is a graph theoretic analysis of political polarization using United States Senate voting data from the 1990s through February 2018. The first function I developed takes a folder containing XML files, fills a graph full of nodes for each Senator, and for each node connects all possible edges with appropriate weights. The next function I developed performs homophily analysis on the graph. Homophily is a measure of how polarized two sets of nodes are on a graph. When it comes to measuring homophily, the idea is to compare the given graph's fraction of heterogeneous edges with that of a graph with the same number of nodes and randomly assigned edges. The procedure for measuring homophily is as follows:

1. Calculate party probabilities p, q
2. Calculate homophily threshold $2pq$
3. If % heterogeneous edges $< 2pq$, then there is evidence for homophily

The difference with our Senate graph is that it is weighted, this procedure only applies to unweighted graphs. In the following section I'll go over how I accounted for this. I made sure and followed this specification and also added the ability to produce a visual plot of each graph with the thickness of each edge corresponding to the edge's weight. I was able to draw both parties in a line, however I think the visualization would be much more effective if each party were two circular subgraphs, then the connectivity between and inside each party would be visible. That being said, observing the density and thickness of the heterogeneous edges still gives a clear indication on the differences between graphs with low and high homophily. Below I have given examples of low, medium, and high homophily to demonstrate the principle.

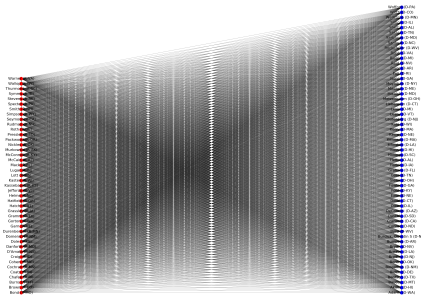

Figure 1: Low homophily

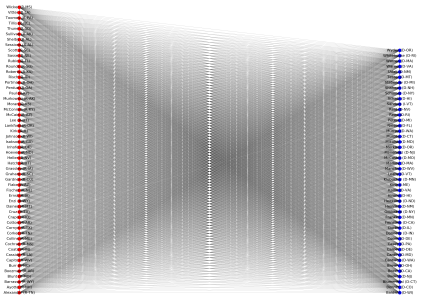

Figure 2: Moderate homophily

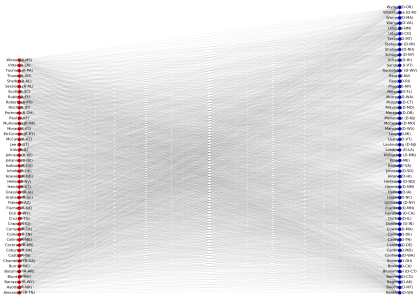

Figure 3: High homophily

As expected, the higher the polarization between parties, the fewer connections between them (corresponding to lighter heterogeneous edges). A graph with no homophily would have equal connectivity throughout.

Before I present my full results, I want to make note of a measure I decided to track and why. Upon inspecting the XML documents I noticed a field that was only in about half of them called "modify date." My intuition tells me votes shouldn't ever be modified, and I haven't been able to find anything else about this online, so I took note of it and made sure to flag if a set of votes had or hadn't been modified. Whether or not this is actually significant is to be determined, however there is a noticeable pattern relating to homophily with the given set of votes.

To measure homophily effectively on this graph, as noted before I needed to convert the weighted graph to an unweighted graph. That process was fairly simple, here it is laid out step by step.

1. Count the number of republicans and democrats and divide by the total number of nodes, p and q.
2. Random heterogeneous edge probability is given by $2pq$.
3. Loop through all edges and add up homogeneous and heterogeneous edge weights.
4. The random "expected" heterogeneous probability is compared with the fraction of "actual" heterogeneous edge weights divided by the combined weights of all edges.

Below are the results for homophily analysis. I also combined folders for similar decades and ran analysis on each decade to provide an overall picture of voting trends over time.

Folder	Homophily Threshold	Cross-Party %	Same-Party % (R/D)		Homophily Rank	Modified?
101	49.406%	44.177%	44.990%	55.009%	Slight	No
102	49.019%	38.463%	36.177%	63.822%	Moderate	No
103	49.280%	39.541%	39.915%	60.084%	Slight	No
104	49.692%	44.646%	61.930%	38.069%	Slight	No
105	49.50%	39.50%	59.458%	40.541%	Slight	No
106	49.406%	33.083%	62.742%	37.257%	Moderate	No
107	49.995%	23.723%	50.703%	49.296%	Strong	No
108	49.98%	22.213%	54.525%	45.474%	Strong	No
109	49.50%	25.502%	58.704%	41.295%	Strong	Yes
110	50%	45.535%	43.991%	56.008%	Slight	No
111	48.558%	27.997%	32.122%	67.877%	Strong	Yes
112	49.82%	34.203%	41.225%	58.774%	Moderate	Yes
113	49.50%	12.802%	42.80%	57.199%	Extreme	Yes
114	49.68%	32.190%	55.761%	44.238%	Moderate	Yes
115	49.877%	26.928%	55.594%	44.045%	Strong	Yes
115-2	49.98%	32.095%	53.972%	46.027%	Moderate	Yes
1990s	49.965%	41.312%	47.907%	52.092%	Slight	No
2000s	49.886%	30.889%	54.249%	45.750%	Moderate	Yes
2010s	49.874%	28.091%	47.295%	52.704%	Strong	Yes

There are a few trends noticeable here to discuss. First I should mention the trend I noticed with the modified votes. In general, the modified votes have a higher level of homophily. Again, this might be totally anomalous, I just thought it was worth mentioning just in case. The main trend to notice is the general trend of polarization increasing over time. If we look into the votes where polarization was higher (Strong level), we can usually find out why as well. In a few folders exhibiting strong homophily (106-109), we find that most of the votes have to do with military spending and issues around the Iraq War. The decade analysis is very telling, showing a steady trend toward increasing polarization.

It can be observed that folder 113 is the most polarized collection of votes. Upon further inspection, I found that a majority of the votes in folder 113 have to do with the Affordable Care Act and other issues around Health Care. This is both a really interesting find and seemingly an intuitive confirmation that our model is functioning properly. Health care in the United States, specifically President Obama's Affordable Care Act, is without a doubt one of the most contentious topics in politics in the last decade.

From my analysis I feel I can safely conclude that the political polarization most of us intuitively feel increasing can be measured directly through graph theoretic homophily analysis. The level of homophily moves from the Slight-Moderate range to the Moderate-Strong range following the election of George W. Bush and seems to

steadily increase from there. One measure I think that would be the most informative would be if we were provided access to a list of political donations to Senators and measured how that effected voting trends.

2 Analysis of Experience vs Cooperation

For this assignment we were asked to come up with another way to measure the data to detect some sort of significant relationship. The metric I decided to track is Senator experience vs. level of cooperation. I wanted to perform some sort of cumulative analysis on the votes, so I combined all the votes into one folder. Keep in mind this measurement is very general and could probably be made more accurate with more specific information. The basic idea is as follows:

1. Every time a Senator participates in a vote, add experience point.
2. All nodes have a cooperation level, +1 is added for each heterogeneous edge weight.
3. Compare the high experience Senators with the low experience Senators to observe the trend of cooperation.

I split up the experience and cooperation levels into four categories. For experience we have [New, Experienced, Very Experienced, and Long Standing] and for cooperation we have [Low, Medium, High, Extreme]. In the table below are my findings.

Cooperation Level/Experience	Low	Medium	High	Extreme
New	42	28	0	0
Experienced	7	105	24	0
Very Experienced	0	3	51	15
Long Standing	0	0	1	3

It can be observed even visually from the chart that there is a general linear trend for Senators to be more and more cooperative the longer they've been in office. Along with that, the majority of Senators are either Low or Medium cooperation, only 33% of Senators reach the "High cooperation" level. Again, this is a very general trend, this could be improved with more specified data such as how many years each Senator has been in office, how many times they've been elected, or what political donors they're affiliated with. The general statistics for the cumulative graph is given below.

```
279 nodes
21671 total edges
10799 edges between Senators of opposing parties

homophily evidence threshold : 49.999\%
cross-party edge weights     : 33.129\%
same-party edge weights      : 66.871\%
    republicans : 49.781\%
    democrats   : 50.219\%
```

We also see here that overall, there is about a 60/30 split between voting similarly or dis similarly against your peers. We can also conclude that our results show fairly strong evidence for general homophily among the Senator network from the last 30 years.

3 Code

```
# -------------------------------------
# Zack Fravel
# 010646947
# CSCE 5014 Applications of graph Theory
# -------------------------------------
```

```
# HW2.py
#
# Analysis of  U.S. Senate roll call
# voting data using NetworkX
# -----------------------------------

# ###################################################################### #
# IMPORTANT: HW2.py must be in the same directory as roll_call_votes  #
# ###################################################################### #

# Program takes a directory of US roll call votes specified by folder number (101 - 115-2).
# The program parses the xml documents and generates a graph where each node
# represents a US senator and each edge corresponds to Senators voting the same way.
# The weight of an edge is the number of times the Senators voted the same way.
# The thickness of the edge on the produced plot is also an indication of cooperation.

import os
import networkx as nx
import matplotlib.pyplot as plt
import xml.etree.ElementTree as et


# Returns the path to all the .xml documents within one folder
def get_xml_path(filepath):

    xml_path_list = []

    for filename in os.listdir(filepath):

        if filename.endswith(".xml"):
            xml_path_list.append(os.path.join(filepath, filename))

    return xml_path_list


# Measure the cooperation of Senators against their experience
# This function is designed to take in a cumulative list of all the votes
def measure_cooperation_vs_experience(g):

    # Create lists to be used in the function
    # Cooperation lists didn't end up being used, could be used to extend further later on
    low_cooperation = []
    low_coop_exp = []
    medium_cooperation = []
    med_coop_exp = []
    high_cooperation = []
    high_coop_exp = []
    extreme_cooperation = []
    extr_coop_exp = []

    # For each node, add the experience for the Senator on each cooperation level
    for nodes in g.nodes():
        # Extreme Cooperation
        if g.node[nodes]['cooperation_level'] > 3000:
            extreme_cooperation.append(g.node[nodes])
            extr_coop_exp.append(g.node[nodes]['experience'])
        # High Cooperation
```

```
    elif 1500 < g.node[nodes]['cooperation_level'] <= 3000:
        high_cooperation.append(g.node[nodes])
        high_coop_exp.append(g.node[nodes]['experience'])
    # Medium Cooperation
    elif 500 < g.node[nodes]['cooperation_level'] <= 1500:
        medium_cooperation.append(g.node[nodes])
        med_coop_exp.append(g.node[nodes]['experience'])
    # Low Cooperation
    elif g.node[nodes]['cooperation_level'] <= 500:
        low_cooperation.append(g.node[nodes]['cooperation_level'])
        low_coop_exp.append(g.node[nodes]['experience'])

# Create experience list
exp_list = [low_coop_exp, med_coop_exp, high_coop_exp, extr_coop_exp]
count = 0

# Loop through each Senator's experience level for each level of cooperation
for l in exp_list:

    new = 0
    experienced = 0
    very_experienced = 0
    long_standing = 0

    # Print cooperation level
    if count == 0:
        print("Low Cooperation")
        count += 1
    elif count == 1:
        print("Medium Cooperation")
        count += 1
    elif count == 2:
        print("High Cooperation")
        count += 1
    elif count == 3:
        print("Extreme Cooperation")

    print()

    # Loop through list and add respective experience levels
    # Experience levels were determined by number of votes participated in
    for e in l:
        if e <= 20:
            new += 1
        elif 20 < e <= 60:
            experienced += 1
        elif 60 < e <= 120:
            very_experienced += 1
        elif e > 120:
            long_standing += 1

    # Print experience levels
    print("New Senators              : ", new)
    print("Experienced Senators      : ", experienced)
    print("Very Experienced Senators : ", very_experienced)
    print("Long Standing Senators    : ", long_standing)
    print()
```

```
    pass


# Determine amount of homophily in a given graph of U.S. Senate votes
def measure_homophily(g):

    # List the number of edges and nodes for verification
    print(g.number_of_nodes(), "nodes")
    print(g.number_of_edges(), "total edges")
    count = 0
    for u, v in g.edges():
        if g[u][v]['opposed'] is True:
            count += 1
    print(count, "edges between Senators of opposing parties", '\n')

    republican_probability = 0
    democrat_probability = 0
    heterogeneous_weight = 0
    homogeneous_r_weight = 0
    homogeneous_d_weight = 0

    for node in g.nodes():
        if g.node[node]['party'] == 'R':
            republican_probability += 1
        else:
            democrat_probability += 1

    # p
    republican_probability = republican_probability / g.number_of_nodes()
    # q
    democrat_probability = democrat_probability / g.number_of_nodes()
    # 2pq
    cooperative_edge_probability = 2 * republican_probability * democrat_probability

    # Loop through edges and begin weighted to unweighted graph conversion process through averaging weigh
    for u,v in g.edges():
        # If edge is opposing, add to heterogeneous weight
        if g[u][v]['opposed'] is True:
            heterogeneous_weight += g[u][v]['weight']
        # If not, add to homogeneous weight
        else:
            # Split between parties for later use?
            if g.node[u]['party'] == 'R':
                homogeneous_r_weight += g[u][v]['weight']
            else:
                homogeneous_d_weight += g[u][v]['weight']

    # Set weights
    homogeneous_weight = homogeneous_r_weight + homogeneous_d_weight
    combined_weight = heterogeneous_weight + homogeneous_weight

    # Set homophily threshold
    homophily_threshold = cooperative_edge_probability
    # Set percentages
    cross_party_percentage = heterogeneous_weight/combined_weight
    same_party_percentage = homogeneous_weight/combined_weight
```

```
    print("homophily evidence threshold :", homophily_threshold)
    print("cross-party percentage       :", cross_party_percentage)
    print("same-party percentage        :", same_party_percentage)
    print("     republican :", homogeneous_r_weight/homogeneous_weight)
    print("     democrat   :", homogeneous_d_weight/homogeneous_weight, '\n')

    # Determine homophily
    if (homophily_threshold - cross_party_percentage) < .1:
        print("There is slight evidence for homophily")
    elif (homophily_threshold - cross_party_percentage) < .2:
        print("There is moderate evidence for homophily")
    elif (homophily_threshold - cross_party_percentage) < .3:
        print("There is strong evidence for homophily")
    elif (homophily_threshold - cross_party_percentage) < .4:
        print("There is extremely strong evidence for homophily")

    # Print homophily percentage
    print(round(homophily_threshold - cross_party_percentage, 5), "%")

    pass


# Add edges for a given senator on a given vote
def add_edges(g, member):

    global edge_width

    # Check all nodes
    for nodes in g.nodes():

        # Compare votes with current node
        if member[5].text == g.node[nodes]['vote']:

            # Edge already exists
            if g.has_edge(member[0].text, nodes):

                # Modify edge attributes (increase thickness and weight)
                g[member[0].text][nodes]['thickness'] += 0.01
                g[member[0].text][nodes]['weight'] += 1

                # If from opposing parties
                if g.node[member[0].text]['party'] != g.node[nodes]['party']:
                    # +1 to cooperation_level on each node
                    g.node[member[0].text]['cooperation_level'] += 1
                    g.node[nodes]['cooperation_level'] += 1

            # Edge doesn't exist
            else:

                # Add Edge
                g.add_edge(member[0].text, nodes, thickness=0.01, weight=1, opposed=False)

                # Indicate opposition if Senators are in different parties
                if g.node[member[0].text]['party'] != g.node[nodes]['party']:
                    g[member[0].text][nodes]['opposed'] = True
                    # Add +1 to cooperation_level on each node
```

```
                    g.node[member[0].text]['cooperation_level'] += 1
                    g.node[nodes]['cooperation_level'] += 1

        # Remove all edges between senators and themselves
        if member[0].text == nodes:
            if g.has_edge(member[0].text, nodes):
                g.remove_edge(member[0].text, nodes)

    # Assign width attribute for each edge to the edge_width list
    for u, v in g.edges():
        edge_width = g[u][v]['thickness']

    pass


# Create a graph given a folder of roll call votes (XML documents)
def initialize_graph(g, folder):

    ry = 0  # Y-location variables for R nodes on plot
    dy = 0  # Y-location variables for D nodes on plot
    modify_flag = False

    # Loop through each roll call vote in each folder
    for file in folder:

        # Set XML file in folder to be parsed
        tree = et.parse(file)
        root = tree.getroot()
        print('\n', file, '\n')

        # Check for root structure
        if root[5].tag == 'modify_date':
            member_level = 17
            modify_flag = True
        else:
            member_level = 16

        # Loop through each senator in roll call vote
        for members in root[member_level]:

            # If node doesn't exist
            if members[0].text not in senators:

                # Add senator to list
                senators.append(members[0].text)
                # Add node
                g.add_node(members[0].text, position=(0, 0), experience=1, party='n/a', vote='n/a',
                           cooperation_level=0)

                # Check for Republican or Democrat
                if members[3].text == 'R':

                    # Set node position
                    g.node[members[0].text]['position'] = (0, ry)
                    # Set node party affiliation
                    g.node[members[0].text]['party'] = 'R'
                    # Set node color
```

```
                    color_map.append('red')
                    # Increment position variable
                    ry += 0.3

                else:

                    # Set node position
                    g.node[members[0].text]['position'] = (1, dy)
                    # Set node party affiliation
                    g.node[members[0].text]['party'] = 'D'
                    # Set node color
                    color_map.append('blue')
                    # Increment position variable
                    dy += 0.3

            # If node already exists
            else:

                # Add experience point
                g.node[members[0].text]['experience'] += 1

            # Record vote and store to node
            if members[5].text == 'Yea':
                g.node[members[0].text]['vote'] = 'Yea'
            elif members[5].text == 'Nay':
                g.node[members[0].text]['vote'] = 'Nay'
            else:
                pass

            # Add edges to node
            add_edges(g, members)

        # Reset vote on each node so as to avoid incorrect edge creation
        for node in g.nodes():
            g.node[node]['vote'] = 'n/a'

    if modify_flag is True:
        print("This analysis contains modified votes")

    pass


# Create lists and graph
g = nx.Graph()      # Create graph
color_map = []      # Colors List for party affiliation
edge_width = []     # Use to record weight of each edge
senators = []       # List of senators

# Ask for user input
requested_folder = input("Enter folder number (101, 102, ... 115, 115-2): ")
folder_string = "roll_call_votes/"+requested_folder

# Initialize graph from the folder specified by the user
initialize_graph(g, get_xml_path(folder_string))

# Measure cooperation vs. experience
# Un-comment to include this in the run
```

```
# measure_cooperation_vs_experience(g)

# Measure homophily
measure_homophily(g)

# Assign positions from node attributes
position = nx.get_node_attributes(g, 'position')

# Set resolution of the plot
plt.figure(dpi=400)

# Draw graph
nx.draw(g, pos=position, node_color=color_map, node_size=8, width=edge_width,
        with_labels=True, font_size=4)

# Show plot
plt.show()
```