CS 6630 Project Journal

Beginning 11-4-2015

Curtis Miller and Jignesh Rawal

Masters of Statistics and Masters of Information Systems

Index

Contents

PoliVis Project Proposal

Entry by Curtis Miller

0.1 Background and Motivation

The U.S. Congress is a complex entity, with 100 senators and 435 members of the House of Representatives, or 535 members total. While political party is certainly a very important factor in terms of how an individual congressman will vote, it is not a purely determining factor as in other countries where parties often vote as blocks and legislators merely occupy a seat for their political party. Congressmen, especially in the Senate, often act individually, and taking a position contrary to the party's position is not unheard of (although Congress is becoming more polarized and crossing party lines is less common). Thus, the political structure of Congress is much more complex than simply which party holds the majority.

This project aims to visualize congressional voting patterns in terms of how similarly congressmen vote. If two congressmen tend to vote the same on bills or tend to cosponsor the same legislation, they may be considered "similar". When thinking about relationships between members of Congress in this way, we would like to visualize these relationships. This may be useful to politically active individuals and organizations such as lobbyists or lobbying groups and firms (where I worked as an intern for over a year), where determining a strategy often requires understanding these kinds of relationships.

0.2 Project Objectives

The project's overarching goal is to allow exploratory analysis of congressional relationships. A user of the app we create would be able to select a congressman or group of congressmen (say, a state's congressional delegation, members of a particular committee, or individuals who vote "Yay" in favor of a certain bill). The app would then use some visual idiom to show how that congressman (or the group) relates to the other members of Congress and show how other members are "similar" to the selected individual or group. This could be used to answer a number of questions, such as:

- If an individual congressman is about to sponsor a bill, who are other congressman who may cosponsor it?
- Does a particular delegation (say, the Utah delegation) tend to vote similarly to the delegations from Utah neighbors? In other words, do we see strong regional voting blocks?
- Is Congress becoming more polarized? Do we see less crossover than we once saw before?
- Can we identify any "maveriks"? That is, can we find individuals who are largely marginalized? On the flip side, can we find individuals who vote very similarly to their party or the Congress as a whole?

These are interesting relationships and questions that the app created in this project could discover and answer.

0.3 Data

The Library of Congress website, along with the websites for the U.S. Senate and the House of Representatives, contain voting and sponsorship records. These are usually HTML or XML documents. The roll call votes are XML documents, so they are easier to read and process. Prof. Lex has directed us to a similar project done in a class at Harvard University that may already provide scrapers and data for use.

0.4 Data Processing

The data in the XML files would need to be read and processed. Also, it may be more efficient to process the data and creating the data structures used to calculate "similarity" (which, between two members of Congress, is the probability another member of Congress votes similarly to the selected member, or in another view, sponsor's the same legislation) prior to displaying that data visually. In other words, the relationships likely will not be determined dynamically, but from a file providing the relationships in a way that can be easily processed when the app is online.

0.5 Visualization Design

The visualization design needs to show the chance any legislator votes similarly to the selected legislator. It needs to show this at the regional level and in some other idiom. My visualization includes an (interactive) map and an (interactive) scatter plot, where the x-axis represents congressional members' ideological position (as determined by the DW-NOMINATE score, which measures ideological extremity) and percentage of times of agreement is the y-axis. The x-axis scale is actually an adjustable power scale that allows for the user to make differences appear more or less extreme on the ends, which allows for easier identification of outliers.

The design of the visualization is attached to this document.

0.6 Must-Have Features

The features this app must have include:

- Roll call voting data for the Senate for the 114th Congress.
- A scatter plot representing the percentage of time congressmen vote with the selected individual or group versus their ideological score.
- A map that shows via luminosity how frequently a congressman (or group) agrees with state delegations across the country, with hue indicating with which party this agreement frequently occurs (it is a continuous scale between blue and red).
- The ability to select a congressman from a side list, the scatter plot, or a congressional delegation from the map.

0.7 Optional Features

Optional features include:

- Roll call voting data for the House of Representatives (this is more complex because the House has many more members).
- Ability to look at patterns for different Congresses (in other words, the ability to look back in time).
- A power scale slider that allows for exageration of differences, making outliers easier to spot.

- Name, state, party, committee, tenure, and bio information for a selected legislator.
- When no one is selected, the visualization shows who is often voting on the winning side of issues.
- An additional visualization that shows who frequently cosponsors legislation sponsored by the selected congressman (this is difficult to do because the data is difficult to obtain, requiring scraping, and it is difficult to apply this when more than one congressman is selected).

0.8 Project Schedule

The data should be collected and processed by November 6th. The visualizations should be in a working state by November 20th. Sliders and selectivity should be completed by November 27th, and the overall app structure should be completed by December 4th. At this point, the app will be completed and ready to be deployed. Optional features may be added after the required features are present.

Wednesday, 4 November 2015

Entry by Curtis Miller

We have made some progress on downloading the data and are a step closer to making the data useable. I would like to move forward on soon being able to making a useable JSON file. Another thing we could do today is create some of the files (HTML, JSON, JSS, etc.) that will compose the final visualization, along with files that, while not a part of the final project, will allow us to test whether certain components are working. For example, one test HTML file will load the data. Another may perform some of the calculations and processing necessary in the final visualization. We may add more of these as necessary.

1 Data Acquisiton, Processing, and Preparation

Today we made progress in data acquisition and making the data useable so it may eventually be loaded into the visualization. We have downloaded data and

1.1 Data Acquisition

Jignesh Rawal downloaded the data from the U.S. Senate website and created spreadsheets containing the necessary information. The data exists on the Senate website as XML files, so presumably he downloaded it and somehow parsed it into a spreadsheet. The data on the sheets "vote" and "Vote 296&295" appear to be the most useful at this point in time. The data on "vote" contains the results of votes that will be used for a default display when no Senators have been selected in the visualization. "Vote 296&295" has individual votes on a particular roll-call vote that will be needed. Right now, this is not in a format that I imagine the end data being in, which is effectively a two-dimensional array or matrix, but I could read the data in these sheets into R and would then be able to turn it into a matrix and then the JSON file that I want. However, I am not sure whether JSON would allow the data to be processed like I would in R, so it's possible that a matrix is not the best way to handle it.

1.2 Data Pre-Processing and Preparation

Perhaps a better structure for the end JSON file used in the visualization would be as follows:

```
{
  "senators":[
    {
      "name":"Ayotte",
      "state":"NH",
      "party":"R",
      "votes":[
        "Yea",
        "Nay",
        "Yea",
        "Not Voting",
        ...
        "Yea"
      ]
    },{
      "name":"Alexander",
      ...
    }, ...
    },{
      "name":"Wyden",
      ...
    }
  ],
  "winning_vote":[
    "Yea",  // Not how the result is actually
        recorded (could be "Passed" or "Confirmed")
    "Nay",
    "Nay",
    ...
    "Nay"
  ]
}
```

This data format appears to be a format that would allow for easier computation in JavaScript than if it were a two-dimensional array. When a senator is selected, their object in this data object is added to the object holding all selected senators. Then, the function that computes similarity would iterate through all senators *not* in the selection and compute similarity based on the array `"votes"`.

Getting the data from the spreadsheet format into a format like this would require loading the data into `R`. Hopefully I can write an `R` function soon that does this, while Jignesh Rawal gets more data and processes it.

Tuesday, 10 November 2015

Entry by Curtis Miller

1 Data Acquisiton, Processing, and Preparation

1.1 Data Pre-Processing and Preparation

Jignesh Rawal has loaded most of the data into the online spreadsheets. Now the data must be processed and turned into a useable format. Since I know R, I will try to use it to create a JSON file with the data in the above format.

I use the following R code.

```r
# install.packages("rjson")
library(rjson)

# Read in Senate voting record; based on the sheet Vote 296&295
read.csv("SenateVote114.csv") -> senate.vote
attach(senate.vote)
# Constructing unique name, state, and party pairing
senators <- last_name[1:100]
home_state <- state[1:100]
party_member <- party[1:100]

# Getting votes on issues
record <- t(sapply(senators, function (sen) {
  return(sapply(unique(vote_number), function (vote) {
    return(vote_cast[which(last_name == sen & vote_number == vote)])
  }))
}))

# Create list data object that will become the JSON string
data.obj <- lapply(1:100, function (i) {
  return(list(
    name = senators[i],
    state = home_state[i],
    party = party_member[i],
    votes = record[i,]
    ))
})
names(data.obj) <- senators

# Create metadata
meta.obj <- list("senators" = senators, "votes" = unique(vote_number))
detach(senate.vote)

# Read in results of votes; based on sheet vote
read.csv("SenateResults114.csv") -> senate.res
attach(senate.res)
levels(result) <- c("Yea", "Yea", "Yea", "Nay", "Yea")
data.obj$winning_vote = result

# Convert to JSON string
```

```
toJSON(data.obj) -> json.string
toJSON(meta.obj) -> json.meta.string

# Save string as json file
json.file <- file("SenateRecord114.json")
write(json.string, json.file)
close(json.file)

json.file <- file("Senate114Metadata.json")
write(json.meta.string, json.file)
close(json.file)
```

This code produces the JSON file that is actually used in the applet, along with associated metadata (senators, states, parties, votes). In the process, the structure of the data changed somewhat from what I originally envisioned. Below I describe the new JSON structure.

```
{
  "senators":[
    "Ayotte":{
      "name":"Ayotte",
      "state":"NH",
      "party":"R",
      "votes":[
        "Yea",
        "Nay",
        "Yea",
        "Not Voting",
        ...
        "Yea"
      ]
    },
    "Alexander":{
      "name":"Alexander",
      ...
    }, ...
    },
    "Wyden":{
      "name":"Wyden",
      ...
    }
  ],
  "winning_vote":[
    "Yea",  // Not how the result is actually
        recorded (could be "Passed" or "Confirmed")
    "Nay",
    "Nay",
    ...
    "Nay"
  ]
```

```
}
```

2 Application Interface

I have created the files that will basically serve as a template going forward. I downloaded the HTML5 Boilerplate to use as a start for the site, and going forward we can build off these templates to create the applet.

Jignesh Rawal has started to work on creating the basic JavaScript functions for loading in the data. He unfortunately cannot use git, so he will send me the functions (or at least their basic logic) via e-mail, and I will add them to the git repository where the files are being kept.

Once the data is loaded in, I can start working on functions that perform the necessary computations on the data that will be displayed. If I start by making these functions create dummy data when called, Jignesh Rawal and I can fork responsibilities; he can start working on creating the visualizations (or a basic framework) while I work on the code that gets the actual calculations from the data.

Monday, 23 November 2015

Entry by Curtis Miller

1 Data Acquisiton, Processing, and Preparation

I have made changes to how the data was being processed. This is entirely pre-processing done in R, modifying the structure of the JSON data files that hold the data along with what information they contain. We still need to collect *all* vote records (currently we only have ten votes), and until this is done it is difficult to see what problems will arise (or, for that matter, what problems will go away; read on for more). Some future modifications to the metadata may be made, but otherwise the data is in the desired structure. When we do actually have all the vote data, it will be easy to process it since the logic will not change.

1.1 Data Pre-Processing and Preparation

After loading and interacting with the data, along with trying to learn more about the DW-Nominate method for determining political ideology, I decided that changes needed to be made to how the data was being prepared.

First, since the score for political ideology is basically fixed in the vis, and since I was not finding conclusive information as to how exactly it is computed, I decided that this should be a quantity precomputed in R. This would likely also save time when the visualization was running.

Second, when the data was loaded and live, I decided that the format the data took in the JSON file was not satisfactory; there should be two separate lists, with one containing the members and the other containing the results of the votes. This was important when attaching and reading the data in the visualization.

The metadata was modified to include a list of all states. While I do not believe that this list is currently being used, it likely will be when the map visualization is fully functional.

2 Computation and Data Post-Processing

Jignesh Rawal was struggling to load the data into the app so I had to do this myself. `main.js` loads the data and metadata and assigns it to its position in the `congress` object.

The `congress` object is responsible for handling all computations with the data. It holds the data and metadata, contains the methods for manipulating the selections, and also contains the methods for computing voting similarity. It contains the following methods and attributes:

`congress.selectedMembers` A set containing the senators that are in the current selection.

`congress.data` An object containing data about a Congress and votes, loaded from a JSON file in `main.js`.

`congress.metaData` An object containing congressional vote metadata (senators, votes, etc.) loaded in from a JSON file in `main.js`.

`congress.nonselectedData` Like `congress.selectedData`, but for those members who are not in the selection.

`congress.addMember([member1, ...])` A method that adds members to `congress.selectedMembers` and removes the corresponding members from `nonselectedMembers`. It is preferable to use this method in order to ensure that valid changes to `congress.selectedMembers` are made (such as they are, in fact, a member, or are not already in the selection) while ensuring that they are removed from `congress.nonselectedMembers`. It can take a string of a member's name, or a list of strings of members' names.

`congress.clearMembers()` A function that clears `congress.selectedMembers` of all entries, and adds all members to `congress.nonselectedMembers`.

`congress.memberAgreementPercent` An object of key-value pairs that contains the percentage of times that members who are not in `congress.selectedMembers` voted the same as those who are in `congress.selectedMembers`. This is manipulated by `congress.getAgreementPercent`.

`congress.getAgreementPercent()` This function calculates how frequently members not in the selection voted with those who are in the selection. It manipulates the `memberAgreementPercent` object, where these values are stored.

Every method should give a developer a means to safely manipulate the `congress` object and obtain data from it. It is defined in the `congress.js` file.

2.1 DW-Nominate Calculation

Computing the DW-Nominate score turned out to be much more involved than I initially thought. In short, I was finding very little information about how it is actually computed. It took a while for me to realize that the method basically amounts to dimensionality reduction. Each member of the legislature is thought to be some "distance" away from other members (I defined this "distance" to be the number of votes on which the two members explicitly disagreed) and each member is represented by a vector where each coordinate corresponds to some member in the legislature and the value of some coordinate is the "distance" between the member and the other member corresponding to that coordinate (so there will be one coordinate equal to zero, which is the member's own coordinate; the member never disagrees with herself).

After using this mindset to create a distance matrix, I applied dimensionality reduction to compute a measure for ideology. A negative value corresponds to left-wing political ideology, a positive to right-wing political ideology, and 0 to centrism. This is not exactly identical to the measure of ideology used by DW-Nominate methods; they try to determine what the members' "preferred" ideology is and apply a number of assumptions that I have not been able to translate into computation. Thus our measure of ideology is likely slightly cruder than those actually employed. Nevertheless, it seems adequate, and captures the essence of the Nominate methods.

Since these measures of ideology do not change, they were pre-computed in `R` and loaded into the vis. This likely saves processing time (and helps skirt my weakness in programming a dimensionality reduction algorithm).

2.2 Calculation of Voting Similarity

The `congress` object contains the function that actually computes similarity, which is the proportion of times members voted with the members in the selection. When `selectedMembers` contains only one element, this is fairly straight forward to understand; this is the proportion of times a member voted "Yea" or "Nay" with the selected member whenever the selected member actually voted (so the times that the member did not vote do not count). In the case of an empty set or a set with multiple members, this is more difficult to understand. For multiple members, I define similarity as the proportion of times a member voted the same as *all* members in the selection when all selected members voted the same. If they never voted the same, then this proportion is zero, by assignment. If no members are selected, I instead look at the proportion of times a member voted with the winning coalition of a vote, which is a completely different interpretation of the vis but an interesting one nonetheless, and not an unreasonable default when no members are selected.

All of these computations are handled by the appropriate methods in the `congress` object, and happen automatically whenever the selection is changed.

2.3 Data Selection and Processing

Computing the selection is simple, and is described above. Only the methods included with the `congress` object are allowed to change the selection; this is to help prevent nasty, difficult-to-track bugs. Nothing in the object should be manipulated directly.

Admittedly, I had to decide how I wanted multiple members in a selection or an empty selection handled. They may be somewhat arbitrary solutions to the problems we have had to overcome, but I believe them reasonable ones and unlikely to cause controversy.

3 Data Visualization

One of our visualizations, the scatter plot, is complete and should be ready for deployment in the final vis. The other (admittedly more complicated) vis idiom, the map, is still a work in progress. The good news is that many of the systems for handling these views and coordinating them are ready, so deploying the map should be simple.

3.1 U.S. Map Visualization

Jignesh Rawal has been working on a map visualization for the data. I have not explored what he has done so far in great depth. With the scatter plot completed, the data loaded, an event handler functioning, and other important systems in place, I feel that I should take over development of the map to ensure that it works with the existing system and integrates well. I have asked Jignesh to send me what code he has on the map Saturday night, but I have yet to hear from him.

3.2 Scatterplot Visualization

I have completed the scatter plot visualization, and it has all the features I believe it should have. It is fully interactive and connected to an event handler. In my opinion, it is ready for deployment. Figure 1 shows a screenshot of the scatter plot (in the test HTML file).

The vis has its own object, defined in `scatterVis.js`. In `main.js`, an instance of this object is created after the data is loaded, the `congress` object has been initialized, and the event handler is initialized. It requires that a div with id `scatterVis` be present, along with a `svg` element inside; this is so that the object can be placed where we desire. When the vis is created, the command that creates it must be initialized with values for width, height, and margins, thus making the vis more flexible (an admitted weakness is that the size of the points is not set; we may add this functionality if we feel it is needed).

Most of the methods and attributes of the object are relevant only to the object and are unlikely to be useful to anything outside of that object. The one method that other parts of the app may call repeatedly is the `ScatterVis.update()` method. This updates the data in the vis and displays it. Transitioning is complete with animations. However, even then, since this vis will be linked to other parts of the app, only the event handler `dispatch` will ever call `ScatterVis.update()`, when the `dispatch.selectionChanged()` event is fired. (The map vis will likely update the same way.)

The scatter plot is interactive. When the user hovers the mouse over a point, the point enlarges and reduces opacity, along with bringing a tooltip (a div created with the object) nearby displaying information about the member. When clicked, the current selection is cleared, and the member clicked is added. The `dispatch.selectionChanged()` is notified once the selection has changed, and the vis is updated, complete with animated transitions. Members in the current selection shrink to a radius of zero (rendering the data point representing them invisible), and members returning to the group that is not selected have their radius increased to full size.

There are axes for the data, with the x-axis representing ideological leaning, and the y-axis agreement with members in the selection. There are tick marks on the y-axis, but not the x-axis. This is not an oversight; I do not want users reading too deeply into the numerical value associated with ideology (which may have an ordinal meaning, but not a cardinal one) and rather only see that some members tend to vote on the left or right of the political spectrum.

The visualization is not perfect. The scatter plot can hang around when a transition is made, and transitions can be interrupted if the user is fast. More serious, though, is the fact that right now some members completely overlap and hide others; in other words, some members have identical voting records so there is no way to separate the points being plotted. This is likely because only ten votes are currently used to compute the data, thus making an identical voting record more likely. While

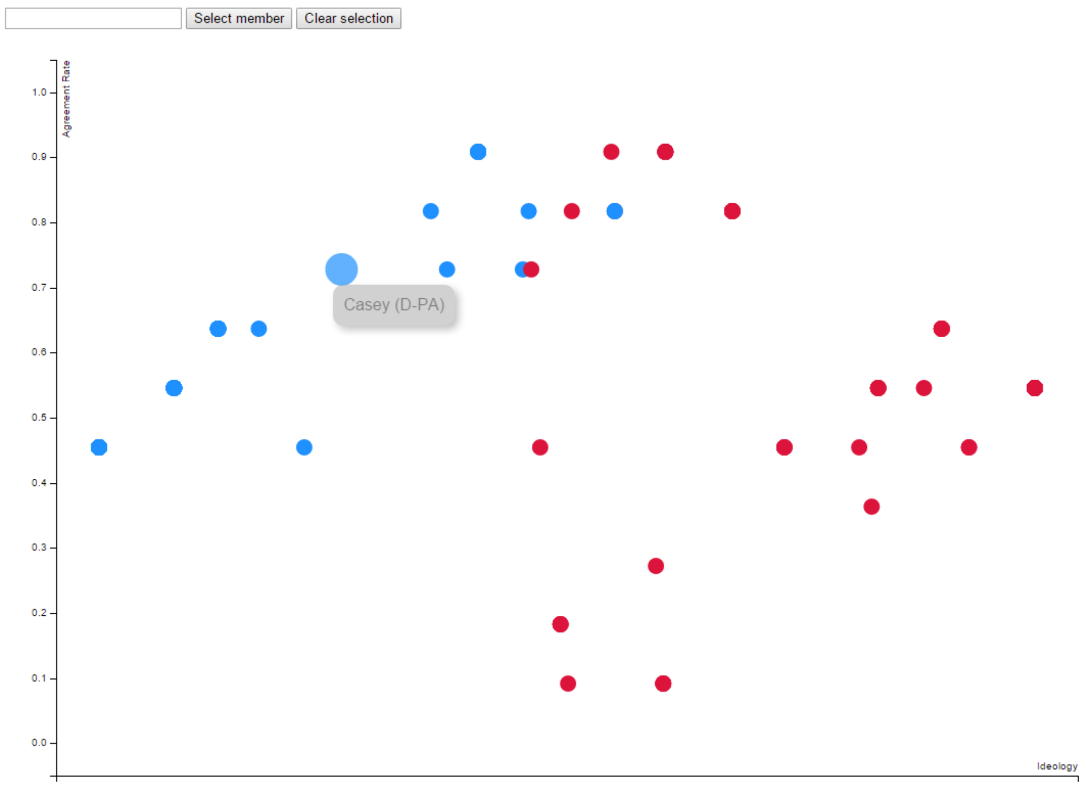

Figure 1: *Screenshot of scatter plot visualization* `scatterVis`

I believe that this will not be an issue when all the data is being used, I am still concerned about this from a design perspective and would like a method that allows for points "covered up" by others to be seen. This is a problem that can be tackled later, though.

3.3 Multiview Coordination between Map and Scatterplot

So far, I have done well to ensure that all objects are able to communicate effectively. The main hub of coordination is the `dispatch` object. This object, at this point, contains only one listener, `selectionChanged`. When called, all views update (so this does not change the selection itself, but should be called after a change has been made). Since the `scatterVis` view is the only one functioning right now, it is the only one that updates, but the map view will also update when it is completed. `scatterVis` and some other HTML input methods (such as buttons and a text box in the test page for `scatterVis`, `scatterplot_test.html`) currently call the method after changing the selection.

Currently, a user can change the selection completely or clear it. Users cannot add to an extant selection. I do not plan on this being something that can be done with `scatterVis` or the map visualization, but some of the other UI elements may be able to do so.

4 Application Interface

The user interface currently is limited to interactivity in the visualizations. I have suggested that Jignesh Rawal look into developing the UI more while I work on creating the visualizations that will be a part of the page. We plan to drop the list of members in favor of a text box with autocomplete functionality. I also believe that we need some way to show what members are currently selected. Admittedly, though, right now I am not too concerned about the UI, and I would like to see what Jignesh Rawal does with it.

4.1 Visualization Interactivity

At this phase, the UI is limited to interactivity in the visualizations and some added HTML input methods that are a part of the test page for `scatterVis`, `scatterplot_test.html` (it consists of a text box where names can be entered, a button to change the selection to the individual listed, and a button that clears the current selection). More interface elements may be added in the final page.

Sunday, 29 November 2015

Entry by Curtis Miller

The major highlight of this entry is the completion ot the map visualization. This required some thought about what the vis idiom actually represents, but I am mostly satisfied with the result. From here, I believe that while Jignesh Rawal works on collecting data, I will work on finalizing the interface, and the project will be largely complete.

1 Data Acquisiton, Processing, and Preparation

There is still a lot of data that needs to be downloaded. We have more votes than previously, but there are still a lot of votes that need to be added. The good news is that the visualizations seem to work without them, but until we have the votes, we cannot know how exactly the visualization will appear.

1.1 Data Acquisition

I received a text from Jignesh Rawal this morning saying that he has added more data. We now have votes 277-307. While it is good that we have 30 votes rather than 10, I think we need to stop adding new votes and try to add votes already cast.

1.2 Data Pre-Processing and Preparation

To make selection of state delegations easier, I added a table to the metadata file containing each state's delegations. This allows the map visualization to easily add delegations, and for the congress object to compute agreement with a state's delegation (more on that later). I also had to add tables that allowed for conversion between a state's abbreviation (for example, "UT") and its full name ("Utah", in this case); this is because the geoJSON recognizes states by their full name, while the methods employed by the `congress` object recognizes states by their abbreviation.

2 Computation and Data Post-Processing

The primary task in terms of computation was deciding what information the map visualization would encode. This is somewhat of a philosophical question that require thought about what I wanted a user to be able to discover with the visualization.

2.1 Calculation of Voting Similarity

A lot of the work done since the last entry was on the map visualization. I wanted the lightness of a state to indicate the agreement of that state's delegation with the members in the selection. However, this forced me to think about what exactly that means. Initially, I thought that a state's agreement would amount to an average of every member in the delegation's proportion of voting with the delegation, but when I thought more about the issue I realized this did not make sense.

I began to think that what I wanted the lightness of the state to indicate was $\mathbb{P}$ (delegation agrees with selection), which the average of the member's agreement does not really capture. However, this probability depends on what "agreement" means. Do both members of the delegation have to vote the same way as the selection, or does only one member have to do so; another way to think of this is, should we represent $x_1 \vee x_2$ or $x_1 \wedge x_2$? The first case is more inclusive and an easier condition for any vote to meet, while the second case is more exclusive and a more difficult condition for any vote to meet.

In my mind, if the $x_1 \wedge x_2$ condition is met, there may be some underlying state issue at stake, and agreement with the delegation indicates that the selection aligns with the interests of the entire state. $x_1 \vee x_2$, on the other hand, indicates whether the selection agrees with either member of the delegation, and so would be more useful to discover new members that agree with the selection. I believe that the end goal of this visualization is to allow users to discover congressmen that vote similarly to some member or coalition, and $x_1 \vee x_2$ is more useful for achieving this end goal. Thus, this was the definition of "agreement" I went with.

I added a list to the `congress` object, `stateAgreementPercent`, that represents

$$\mathbb{P}\left(\text{a member of state's delegation agrees with selection}\right)$$

for each state. This is computed by the `getAgreementPercent()` method attached to the `congress` object. This is the data that is represented by the map visualization.

3 Data Visualization

The major accomplishment since the last entry is the completion of the map visualization. With its completion, the visualization part of the project is effectively complete. The visualizations are already interactive and communicate with one another, so I feel that soon we can complete the project.

3.1 Scatterplot Visualization

I made one change to the scatter plot visualization object creation function `ScatterVis`. It takes an additional parameter, `dotScale`, that scales the dots in the visualization (larger values create bigger dots). This is to help make scaling the visualization easier.

The tooltip `div` has also had its `id` changed to `scatterVisTooltip`, to avoid conflicts with the map tooltip `div`. The text of the tooltip has changed; it shows the percentage of times the member being hovered over agrees with the selection, in addition to the information already present.

3.2 U.S. Map Visualization

This week, I completed the map visualization completely (Jignesh Rawal started work on it, but I had to make a lot of changes to his code to make the visualization work as desired). It is fully interactive, displays all information we desire, has transitions, and is linked to the event handler. I initially programmed it in its own separate environment, like I did with the scatter plot visualization. Adding it to the main page `index.html`, though, was extremely simple. A screenshot of the visualization is shown in Figure 2.

Creation of the map visualization was very similar to creation of the scatter plot visualization. There is an object called `mapVis` that creates and controls the visualization, including loading in the geoJSON file that defines the map (which is the Albers USA projection). This object is described in `mapVis.js`, where the function that creates an instance of a `MapVis` object is programmed. This function requires that there be a `div` element with `id` `mapVis` in the DOM, along with the geoJSON file `us-states.json` in the `data` directory. This function takes the following arguments:

`w`, `h` The width and height of the visualization, respectively.

`mt`, `mb`, `ml`, `mr` The top, bottom, left, and right margins of the SVG, respectively

`scale` The scale of the map, which is the same scale used for `d3.geo.albersUSA()` which creates the map (in other words, it defines distance between points).

There is only one method that an outside user of this object would expect to use, and that is `mapVis.update()`, which is defined `after` the geoJSON file defining the map is loaded. Again, this is likely to be used only by the event handler, when a change in the selection is made.

Information on the map is encoded in three channels: position and shape (for states; represents a delegation), lightness (indicates delegation's agreement with the selection), and hue (indicates the political affiliation of the delegation's agreement with the selection). Also, if a member of a state's delegation is included in the selection, the state's borders are thicker and colored bright green.

The position/shape channel is self-explanatory. I described agreement with the selection earlier. The lighter a state's color, the less the state's delegation agrees with the selection. Remember that this counts *only* for delegation members *not* in the selection; the implication of this is that if both members of the state's delegation are in the selection, the state will appear pure white in the visualization since neither member is not in the selection.

Hue requires more explanation. In the simple case where both members of a state's delegation are members of the same political party, the hue will simply be the hue associated with this party (crimson for Republicans, dodger blue for Democrats, gold for independents). However, there are states where the members have different political affiliations. In this case, the hues are blended depending on which member more strongly agrees with the selection. So in the case where one member is Republican and the other Democrat, if the Republican agrees more with the selection, the hue will be a reddish-purple (assuming the Democrat agrees at all; if the Democrat never agrees, then the hue will simply be red), while if the Democrat agrees more with the selection, the hue will be a blueish-purple.

Color blending is subtractive, which took effort to implement. The RGB color model is an additive color model, while the CMYK color model is subtractive. I had to add a method to the `mapVis` object, `stateColor(state)`, that takes a string representing a state's abbreviation, calculates the color mixing according to the CMYK color model, and outputs an RGB color (the conversion is done inside the method as well) as a hex string. This is the color that is rendered (notice that this color accounts for the lightness channel as well as the hue channel).

Transitions are in place. When the selection is changed and the map updated, the states will shift color to their new colors according to the new selection. I am not sure if this feature will remain. The problem is that this conflicts with a transition that

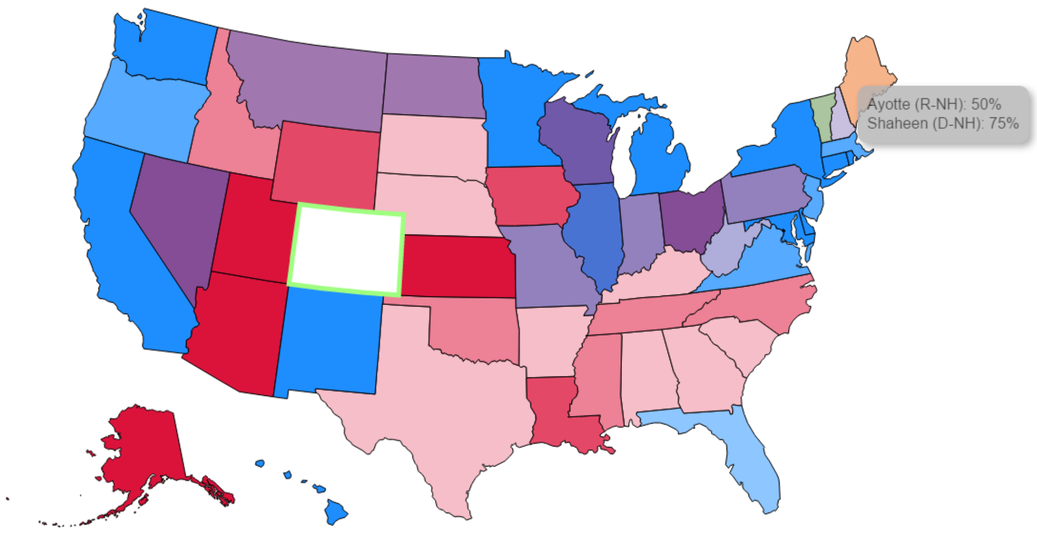

Figure 2: *Screenshot of map visualization* `mapVis`

reduces the opacity of a state's color when the mouse hovers over the state. If the mouse moves mid-transition, the transition is interrupted and can stop mid-change, which is an undesired behavior, especially since the map is interactive.

When the user clicks a state on the map, the selection changes; it is replaced with the members of that state's delegation. This is announced to the event handler, queuing an update of the map (and any other functions listening for the event `selectionChanged`).

Much like the scatter plot, a tooltip appears when a state is hovered over by the mouse. This tooltip shows the members of the delegation, their political party, and the percentage of times they agree with the selection (separately, not jointly).

So far, using the map visualization, I am satisfied with the results. It is not perfect. As mentioned above, there are problems with transitions being interrupted. Another concern is the mixture of the lightness and hue channels; a user may see some hues as darker than others, thus interfering with the lightness channel. The mixture of hues may not be fully intuitive, and the real meaning of "agreement" encoded in lightness may not be intuitive either. However, the tooltips should help a user extract the information they desire and compensate for these imperfections.

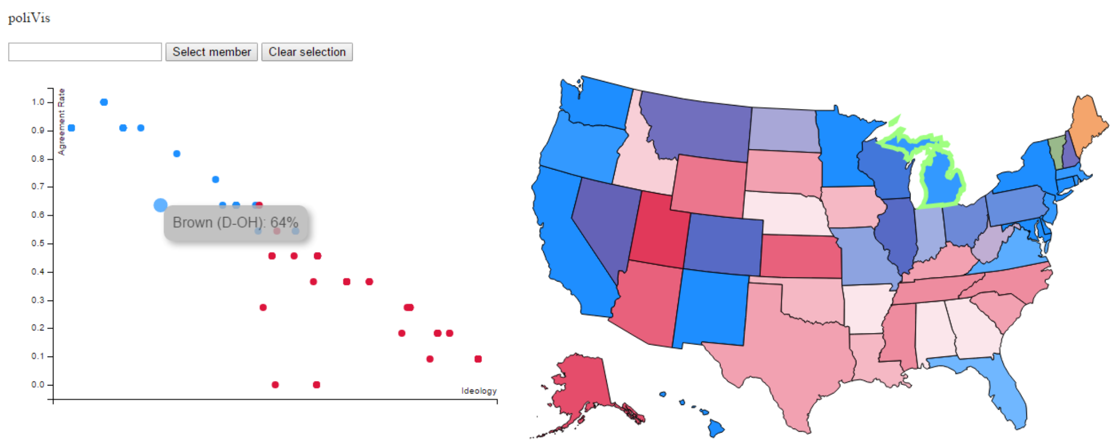

Figure 3: *Screenshot of combined visualizations*

3.3 Multiview Coordination between Map and Scatterplot

From the beginning, the visualizations have been developed to allow for multi-view coordination. An event handler calls the functions that update the visualizations when a change to the selection has been made, and the visualizations have been programmed to use this event handler.

I added the visualizations to `index.html`. A screenshot is shown in Figure 3 They communicate seamlessly. An interesting tendency is that when coalitions (such as delegations) are selected, if both members are from the same party, the visualization looks like a highly ideological member of the Senate.

4 Application Interface

Not much has been done for the interface other than adding the visualizations to the final web page. However, I expect this to be the area that receives the next major push in the coming days, at least by me.

4.1 Visualization Interactivity

The visualizations are fully interactive and present in the final page, `index.html`. At this point in the project, we can add other methods to complete the final webpage, and the majority of the work on this project will be complete. I am thinking we may need to use jQuery at some point, such as when creating an autocomplete textbox.