EXPLORING THE UNIVERSITY OF ARIZONA STUDENT POPULATION THROUGH DATA VISUALIZATION

EXPLORING THE UNIVERSITY OF ARIZONA STUDENT POPULATION THROUGH DATA VISUALIZATION

By

KARYN BAO AN VO

A Thesis Submitted to the Honors College

In Partial Fulfillment of the Bachelors degree
With Honors in

Computer Science

THE UNIVERSITY OF ARIZONA

MAY 2016

Approved by:

Dr. Carlos Scheidegger
Department of Computer Science

Abstract

What makes some students change their majors often, while other students choose a major and stick with it for the entirety of their college life? In order to answer this question, we created a number of interactive data visualizations using data provided by the University of Arizona's office of analytics and institutional research. The programs we wrote allow users to explore the academic careers of the student population at the University of Arizona. Data visualizations are visual representations of a dataset, instead of presenting the user with lists of numbers, visualizations attempt to create renditions of the data that are richer and easier to read. The interactive features we implemented let users filter the dataset, "zooming in" on specific aspects of interest such as demographic features of the dataset. We used the d3.js library to build these visualizations. In this document, we discuss some of the design choices, their consequences, and show some of the findings from this dataset.

Acknowledgements

Special thanks to Dr. Scheidegger for making this all possible. I would also like to thank Dr. Isaacs for a lot of useful resources and helpful input. Thank you Me, Ba, Ong Ba Ngoai, Bo Thuy, and Co Kim for supporting me through college even when I couldn't make up my mind. Thank you Kris for always being there when I need you, and thank you Chi An, Cookie, Hugh, Kien, and Cong for always making my days brighter.

Table of Contents

Abstract 1
Acknowledgements 2
Table of Contents 3
Introduction 4
Materials and Methods 8
- Line Graphs 11
- Tree plot 20
  - Formatting the data 22
  - Building the tree 25
  - Adjusting node spacing 27
  - Adjusting node size 34
  - Attempt 1: height = max.x - min.x of children 35
  - Attempt 2: height = the sum of the heights of all its children 36
  - Attempt 3: height = max.x - min.x + (half of the size of the max shape) + (half of the size of the min shape) 37
  - Attempt 4: Using Attempt 3 to find the height and shifting the nodes so that the top of the parent is the same as the top of the highest child 39
  - Attempt 5: Amending the height and x(y) value of a node 41
  - Adding Ghost Nodes to Max Depth 42
  - Adjusting Height of Tree 43
  - Changing positions of links/text 43
  - Adding a filter based on count of students in each node 44
  - Interactivity 48
- Improving the load time 49
  - Working with Node.js 49
  - Loading JSON files instead of nesting 50

Results/Findings 56
Discussion 66
Conclusion 67
References 68

Introduction

I heard that the average number of times an undergraduate student changes their major is 3 times. For me, it was exactly that amount. In fact, almost everyone I knew had changed their major at least once throughout their college careers. Since everyone's path is different, the question I sought to answer was what the progression of a student that begins in a specific major typically looks like. Having used the University of Arizona directory in the past to reference students, I knew the University of Arizona had information that includes the majors of each student for each semester at minimum. With this information available, we can use the power of programming in order to get a better picture of how people progress through majors based on the majors they begin with.

Understanding large amounts of data has been most recently brought into the public eye in the form of data visualization with the massive amount of data that technology can accumulate today. Putting the student data in the form of an interactive data visualization module is a good idea because being able to visualize the data gives people a better handle on what the data represents and what can be presented to them [8]. To be able to interact with the data allows for users to experiment with different views and representations that can raise new questions and observations [8]. Using computer programming to build this representation gives us the ability to create multiple visualizations quickly and dynamically rather than generating each graph manually or by hand [8]. Visualization "exploit[s] the human visual system as a means of communication" and spotting interesting things in the data is enabled through this.

The inspiration for the project stems from the New York Times article, "Where We Came From and Where We Went, State by State" [1]. The article includes multiple interactive visualization modules in the form of a line graph that displays the states where people move to and where people are from for each state. Users can specify which state they want to examine and which location people are linked to within the graphs along with the time in which they moved.

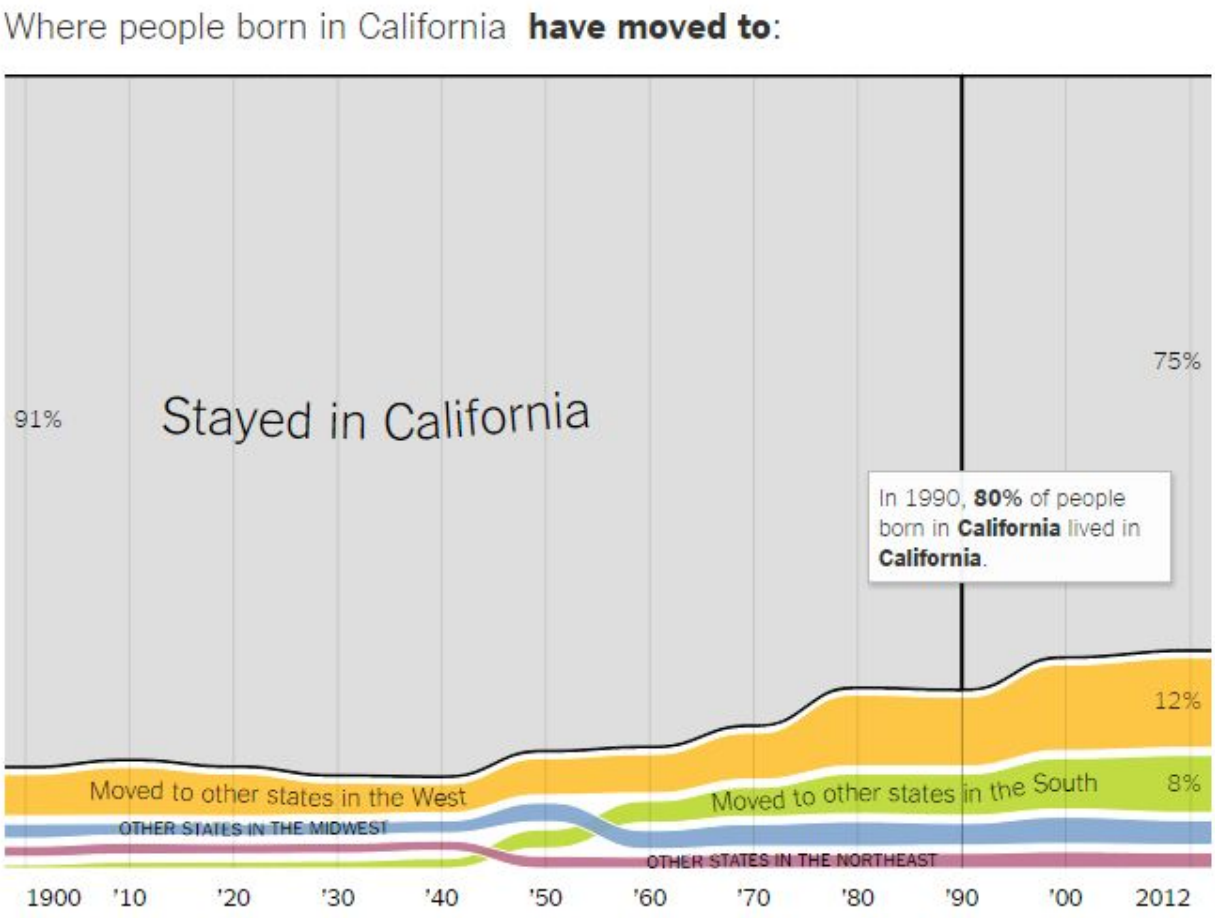

Figure 1. From the New York Times' "Where We Came From and Where We Went, State by State", a view of what migration out of California looked like throughout the years [1].

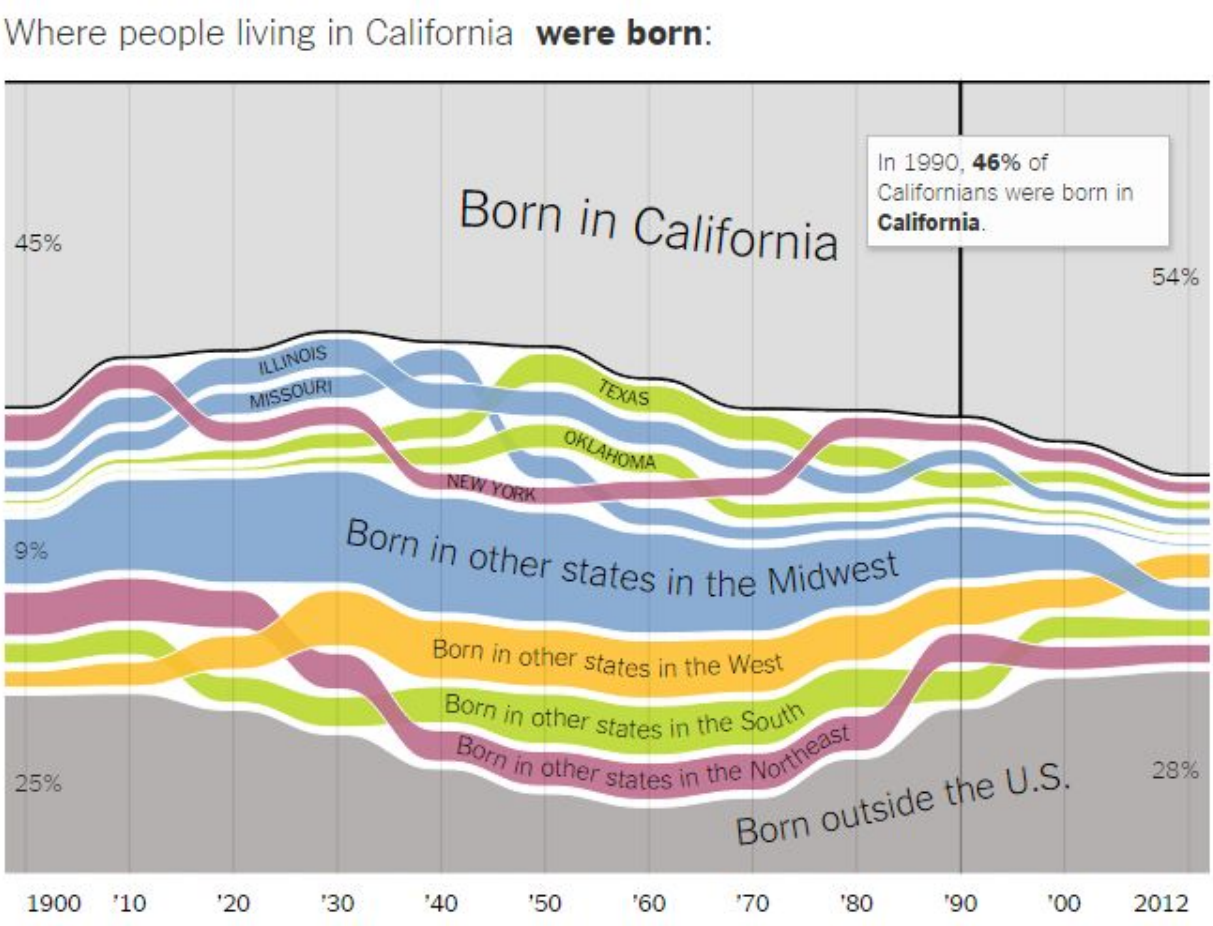

Figure 2. From the New York Times' "Where We Came From and Where We Went, State by State", a view of what migration into California looked like throughout the years [1].

From this, users can get a clear idea of what percentage of people move to a particular region or state just by looking at the graph. Allowing users to select the states they are interested adds to the level of interaction the users have with the data. The data is presented as a snake-like graph that resembles a line graph. Each line represents a state and the thickness of the line corresponds to the number of people relative to the total number of people in the dataset at that point in time. Being able to see how these proportions change over time is also represented since this is a line graph. The colors allow each component to stand out appropriately. The similarities between the question of how people move between states and the question of how undergraduates move between majors made me choose this article as my inspiration.

With the end goal of allowing others to explore this project, using the d3.js library as a tool to create the project was optimal. From the d3 webpage, "D3.js is a JavaScript library for manipulating documents based on data" [3]. Being built specifically for interactive data visualizations, d3 aligned with my goals for the project with its notable functions for processing and manipulating large amounts of data in multiple different formats.

Materials and Methods

Dr. Scheidegger and I made contact with Dr. Guillermo Uribe from Institutional Research at University of Arizona who generously provided a sampling of the UA student data in CSV file format. The final dataset we ended up using was a merging of two datasets that we received with additional and different attributes listed in each record along with some similar attributes. The first set included the semester, student ID, academic plan type, academic plan, and plan description from Fall 2010 to Spring 2015. The second set included the semester, student ID, academic plan type, academic career code, ethnicity, gender, age, and residency status also from Fall 2010 to Spring 2015.

```
STRM,PERSON_SID,acad_plan_type,acad_plan,plan_descr
2104,100001,MAJ,PHLDPH,Public Health
2104,100002,MAJ,SCPSPHD,School Psychology
2104,100008,MAJ,TTEPHD,Teaching & Teacher Education
2104,100009,MAJ,SPANPHD,Spanish
2104,100017,MAJ,CRTVMFA,Creative Writing
2104,100021,MAJ,PSYCBA,Psychology
2104,100022,MAJ,RCTPHD,"Rhetoric, Comp & Teach English"
2104,100029,MAJ,NDSNDG,Nondegree Seeking
2104,100032,MAJ,TTEPHD,Teaching & Teacher Education
2104,100034,MAJ,PHSCPHD,Pharmaceutical Sciences
2104,100040,MAJ,NURSPHD,Nursing
2104,100042,MAJ,PHMYPD,Pharmacy
2104,100048,MAJ,MEDMD,Medicine
2104,100050,MAJ,NDSNDG,Nondegree Seeking
2104,100059,MAJ,MSEBSMSE,Materials Science & Engr
2104,100062,MAJ,CVEBSCVE,Civil Engineering
2104,100065,MAJ,PHMYPD,Pharmacy
2104,100068,MAJ,RCSCBS,Retailing & Consumer Science
2104,100070,MAJ,MATHBA,Mathematics
2104,100076,MAJ,ANTHBA,Anthropology
```

Figure 3. The first data file.

```
"STRM","PERSON_SID","acad_plan","acad_career","Ethnicity Code EPM SP","GENDER_CD","UA_AGE","UA_IPEDS_RESIDENCY
"2104","100001","PHLDPH","GRAD","BLACK","F","30","R"
"2104","100002","SCPSPHD","GRAD","HISPA","F","30","R"
"2104","100008","TTEPHD","GRAD","WHITE","F","51","R"
"2104","100009","SPANPHD","GRAD","HISPA","F","34","R"
"2104","100017","CRTVMFA","GRAD","WHITE","F","33","R"
"2104","100021","PSYCBA","UGRD","HISPA","F","36","N"
"2104","100022","RCTPHD","GRAD","WHITE","F","40","R"
"2104","100029","NDSNDG","GRAD","WHITE","M","35","R"
"2104","100032","TTEPHD","GRAD","WHITE","F","40","R"
"2104","100034","PHSCPHD","GRAD","WHITE","M","36","R"
"2104","100040","NURSPHD","GRAD","WHITE","F","44","N"
"2104","100042","PHMYPD","PHRM","WHITE","M","33","R"
"2104","100048","MEDMD","MEDS","WHITE","F","32","R"
"2104","100050","NDSNDG","GRAD","WHITE","F","33","R"
"2104","100059","MSEBSMSE","UGRD","WHITE","M","22","R"
"2104","100062","CVEBSCVE","UGRD","WHITE","M","22","R"
"2104","100065","PHMYPD","PHRM","ASIAN","F","22","R"
"2104","100068","RCSCBS","UGRD","HISPA","F","22","R"
"2104","100070","MATHBA","UGRD","HISPA","F","22","R"
"2104","100076","ANTHBA","UGRD","HISPA","F","19","R"
"2104","100081","PRARND","UGRD","HISPA","M","19","R"
"2104","100082","GEOSBS","UGRD","HISPA","M","19","R"
"2104","100084","NMSCLASND","UGRD","HISPA","M","19","R"
"2104","100085","NUSCBS","UGRD","WHITE","M","19","R"
"2104","100092","NMSCLASND","UGRD","ASIAN","F","19","R"
"2104","100093","PRPHND","UGRD","WHITE","M","19","R"
"2104","100095","NMSEGND","UGRD","WHITE","M","19","R"
"2104","100096","JOURBA","UGRD","WHITE","M","20","R"
"2104","100097","NMSCLASND","UGRD","WHITE","M","19","R"
"2104","100098","MEEBSMEE","UGRD","WHITE","F","19","R"
"2104","100099","ARCHBAR","UGRD","HISPA","M","20","R"
"2104","100102","NMSCLASND","UGRD","HISPA","F","18","R"
"2104","100106","NMSCLASND","UGRD","HISPA","F","31","R"
"2104","100108","SWESPHD","GRAD","HISPA","F","31","R"
"2104","100115","PHLMPH","GRAD","WHITE","F","27","R"
"2104","100119","PHSCPHD","GRAD","ALIEN","F","39","N"
"2104","100124","ANTHPHD","GRAD","WHITE","F","31","N"
```

Figure 4. The second data file.

The two data files were received at different times, so to aggregate all the fields, we wrote a Java program [4]. They were joined by using a concatenation of the semester, student ID, and academic plan as an identifier for each row. This was done to ensure the correct records were merged in the new data file.

```
STRM,PERSON_SID,acad_plan_type,acad_plan,plan_descr,acad_career,Ethnicity_Code_EPM_SP,GENDER_CD,UA_AGE,UA_IPEDS_RESIDENCY
2104,122310,MAJ,PHYSPHD,Physics,GRAD,ALIEN,M,29,N
2121,667811,MAJ,SPANBA,Spanish,UGRD,HISPA,F,,R
2114,635648,MAJ,HISTBA,History,UGRD,WHITE,M,22,R
2104,648491,MAJ,PRNUND,Pre-Nursing,UGRD,HISPA,F,18,R
2131,914245,MAJ,NMSSCIND,No Major Selected Science,UGRD,WHITE,M,19,N
2111,642671,MAJ,NMSCLASND,No Major Selected Ltr Art Sci,UGRD,HISPA,M,19,R
2144,1128358,MAJ,NURSBSN,Nursing,UGRD,ASIAN,F,22,R
2111,653227,MAJ,MISBSBA,Management Information Systems,UGRD,WHITE,M,21,N
2114,661382,MAJ,POLBA,Political Science,UGRD,WHITE,F,20,R
2114,98343,MAJ,EASPHD,East Asian Studies,GRAD,OTHER,M,32,N
2134,562393,MAJ,NDSNDG,Nondegree Seeking,GRAD,HISPA,F,55,N
2151,873318,MAJ,CHEPHD,Chemical Engineering,GRAD,ALIEN,M,36,N
2151,1133122,MAJ,PRBNND,Pre-Business,UGRD,WHITE,F,18,N
2134,457259,MAJ,NDSNDG,Nondegree Seeking,GRAD,WHITE,F,55,R
2121,753789,MAJ,BIOLBS,Biology,UGRD,HISPA,F,,N
2141,682628,MAJ,BIOLBS,Biology,UGRD,WHITE,M,21,R
2131,900102,MAJ,BIOCBS,Biochemistry,UGRD,HISPA,F,19,N
2114,672726,MAJ,PRBNND,Pre-Business,UGRD,WHITE,M,19,R
2131,657468,MAJ,GERSBA,German Studies,UGRD,WHITE,F,21,R
2114,760854,MAJ,POLBA,Political Science,UGRD,BLACK,M,20,N
2114,806656,MAJ,PRBNND,Pre-Business,UGRD,ASIAN,M,18,R
2151,1199469,MAJ,BNADMBA,Business Administration,GRAD,HISPA,M,29,N
2134,761599,MAJ,SOCBA,Sociology,UGRD,HISPA,F,20,R
2141,682669,MAJ,PSYCBS,Psychology,UGRD,WHITE,M,28,R
2151,754153,MAJ,AEEBSAEE,Aerospace Engineering,UGRD,WHITE,M,21,R
2151,1142300,MAJ,FINBSBA,Finance,UGRD,WHITE,M,22,R
2144,761677,MAJ,ENTRBSBA2,Entrepreneurship,UGRD,ASIAN,M,21,R
2131,673820,MAJ,BNECBSBA,Business Economics,UGRD,WHITE,M,21,R
2121,810852,MAJ,PSYCBS,Psychology,UGRD,WHITE,M,,R
2141,800869,MAJ,PRPND,Pre-Physiology,UGRD,HISPA,F,20,N
2141,656370,MAJ,RCSCBS,Retailing & Consumer Science,UGRD,WHITE,F,22,N
2111,655161,MAJ,MCBBS,Molecular & Cellular Biology,UGRD,WHITE,F,19,R
2121,663143,MAJ,PRPND,Pre-Physiology,UGRD,WHITE,M,,R
2144,759759,MAJ,MKTGBSBA,Marketing,UGRD,WHITE,M,22,N
2134,649568,MAJ,NURSBSN,Nursing,UGRD,WHITE,F,23,R
2134,923214,MAJ,MATHBS,Mathematics,UGRD,WHITE,M,19,N
2111,655203,MAJ,ECONBA,Economics,UGRD,WHITE,M,20,N
```

Figure 5. The new merged data file with post-processing gender data filled in for Spring 2012.

As with most datasets, there were some inconsistencies and missing attributes for some records. For example, no gender or age data was provided for one semester--Spring 2012. As a way around this, we filled in the missing pieces for gender data for that semester using Java. An individual student's gender can be deduced based what their gender was for other semesters. This was assuming that the student in the specific record had data that spanned other semesters and also that the student had not changed gender.

D3.js made it much easier to work with the data since there are built-in functionalities to aggregate and examine each record within the CSV file. One functionality that made it possible for me to obtain the sums of students in a specific major for a specific semester is the nested function. From the D3 wiki, "nesting allows elements in an array to be grouped into a hierarchical tree structure," [7]. So instead of having an array of objects that represent each line in the data, we have this tree structure represented as an array of objects, nested down to the level we specify. Because we wanted the number of students in a specific major for a specific semester, we would nest by major then by semester. Doing so gives us an array of objects whose keys are majors and whose values are an array of objects. This next array of objects within the

values of the major object will have a key that is the semester and a value of an array of student objects with all the attributes of a student.

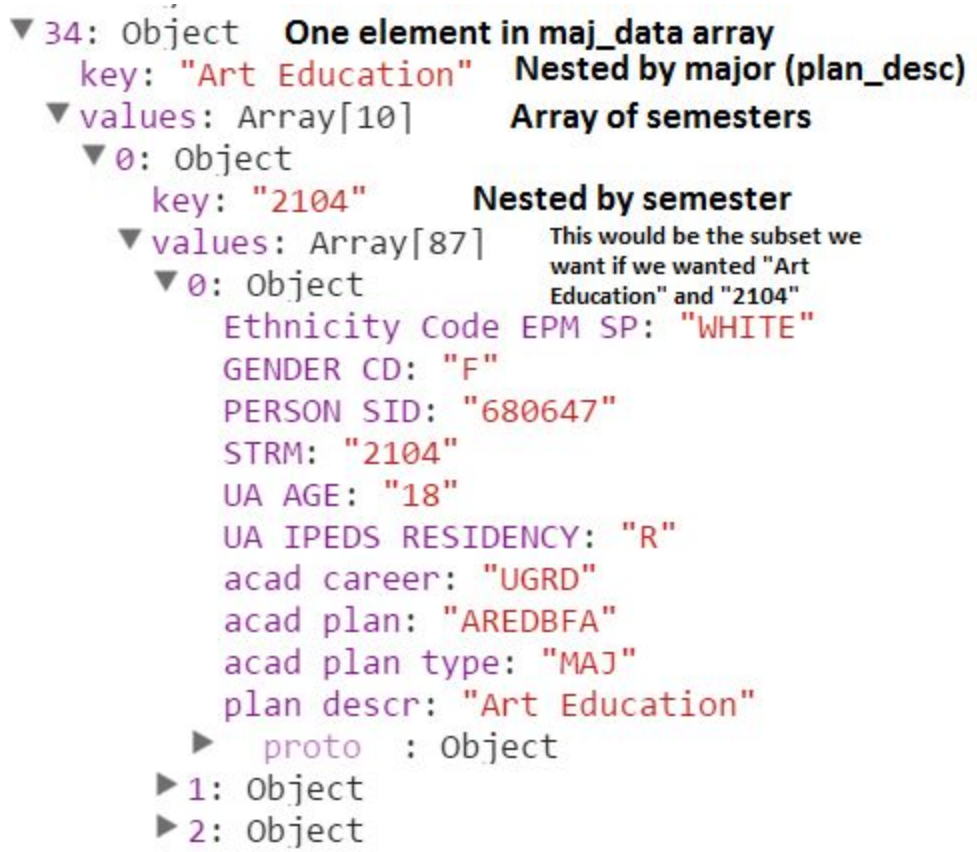

Figure 6. An example of what a nested data structure nested by major then by semester would look like. This was used to give the count of students.

Line Graphs

Using d3's nesting was an integral part of creating the views that would represent the count of students for a given set of restraints. One way to examine the count of students per major per semester is through a line graph, which can directly display how the count of students per semester for each major can change through time and how other majors compare to each other. The goal for the Fall 2015 semester was for me to become familiar with the dataset and the technology we would be working with. As described previously, some pre-processing of the data was required to move forward. Figuring out how d3 could manipulate the data for our interests was the largest learning curve. Creating the plots of the counts of students for each major by demographic attributes was a good way to ease into using the technology while making use of what the dataset could offer us. These plots can tell us what general direction each major is moving towards population-wise compared to other majors.

The initial graph we wanted to create would just be the count of all students for each major for each semester. This does not take into account any additional information of the student like their age, gender, residency status, etc. For the line graph, each line represents a major. The x-axis represents the semesters, and the y-axis tells us the amount of students. All majors were displayed on the same line graph so that one could directly compare the amount of students in one major relative to another.

To get started, we heavily referenced Mike Bostock's Multi-Series Line Chart example to figure out how to create such a visual [7]. This example displays several lines that represent the temperature of different cities throughout different months. The first iteration of the student major line graph used a linear y-axis similar to the example. However, it was clear that a linear y-axis was not fit for our dataset since there were majors with thousands of students and many more with less than 200. This caused the many lines to overlap and be hidden towards the bottom of the graph. Meanwhile, only a couple majors would occupy the top of the graph with large stretches of space. A better way to remedy the skewness towards large values was by adjusting the scale of the y-axis. Instead of having a linearly scaled y-axis, the y-axis would be on a log scale that will increase by a magnitude of 10 [12]. By doing so, the spacing of majors on the y-axis were more evenly distributed since there appear to be a greater number of majors with a lower number of students [2].

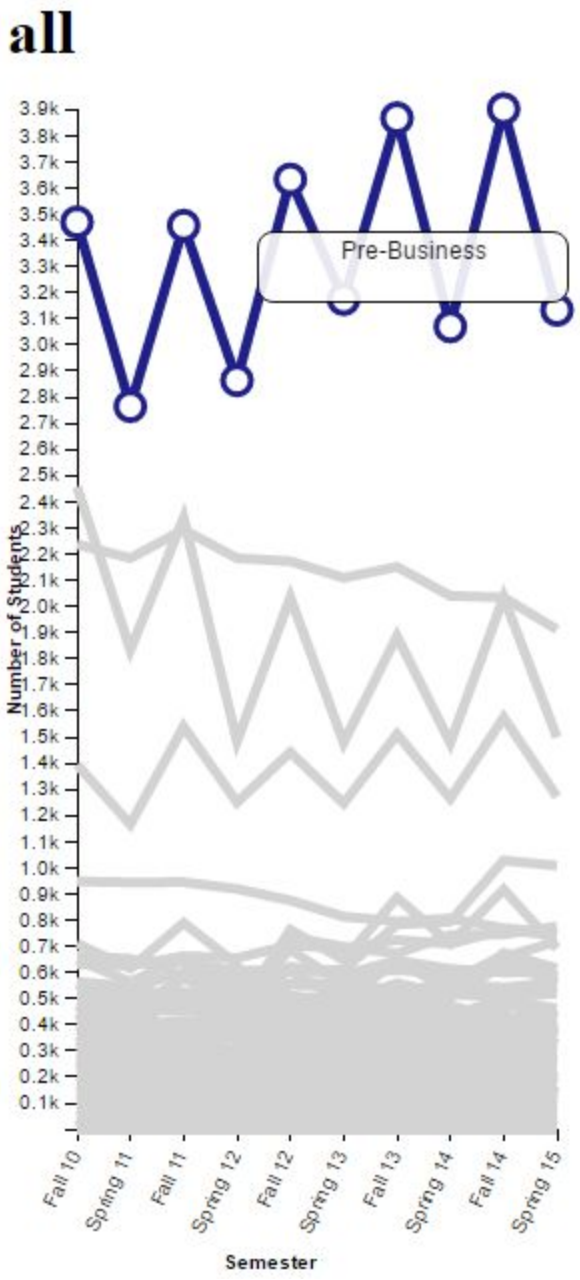

Figure 7. Line graph of all students at the University of Arizona with a linear y-axis. There are only a couple of lines that stand out near the top while the rest are indistinguishable at the bottom.

Another non-optimal characteristic in the first iteration of the line graph were the colors. Similar to the example, the line graph had 10 colors that were specified as a d3 categorical functionality, so each line in the graph was represented with these 10 colors [11].

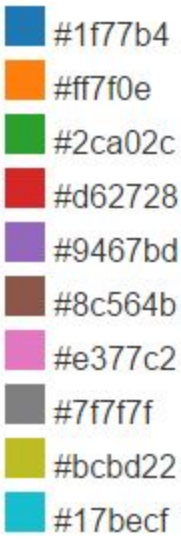

Figure 8. Category 10 colors from d3 [11].

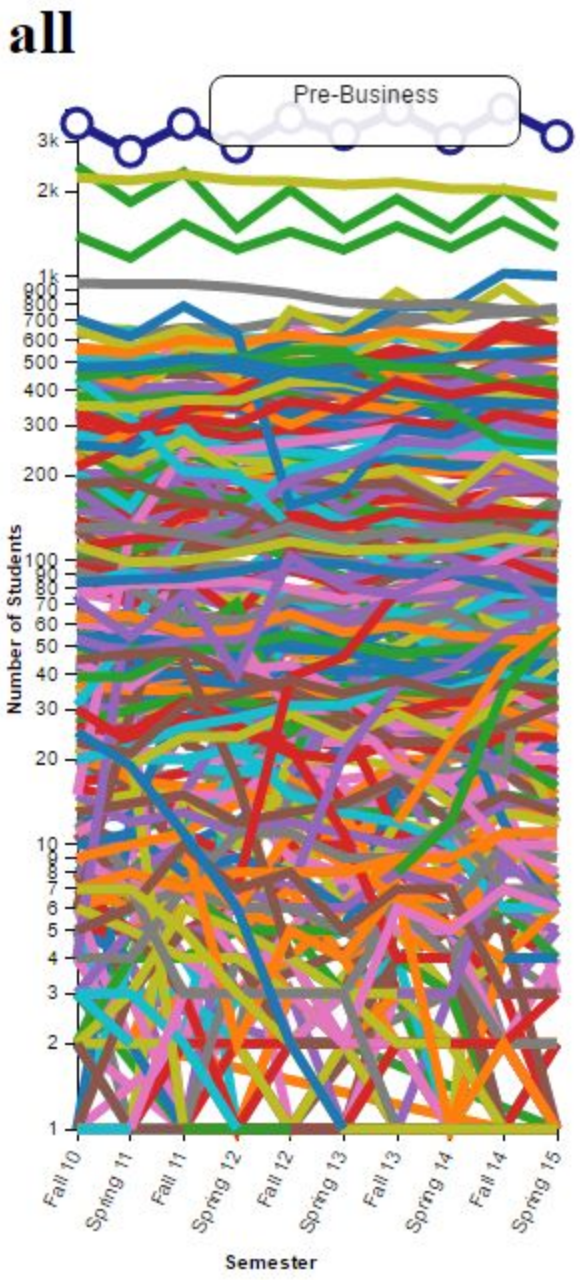

Figure 9. Line graph of all students at the University of Arizona with category 10 coloring and a logarithmic y-axis.

With 371 majors represented within the graph, it was easy to get lost in all the colors and just as simple to lose focus on one particular line. Drawing inspiration from Mike Bostock's Multi-Line Voronoi Graph, all lines were made gray until hovered over and/or selected [6]. This allows for the user to focus on a specific line more easily with the gray lines being more muted and the colored lines more eye-catching.

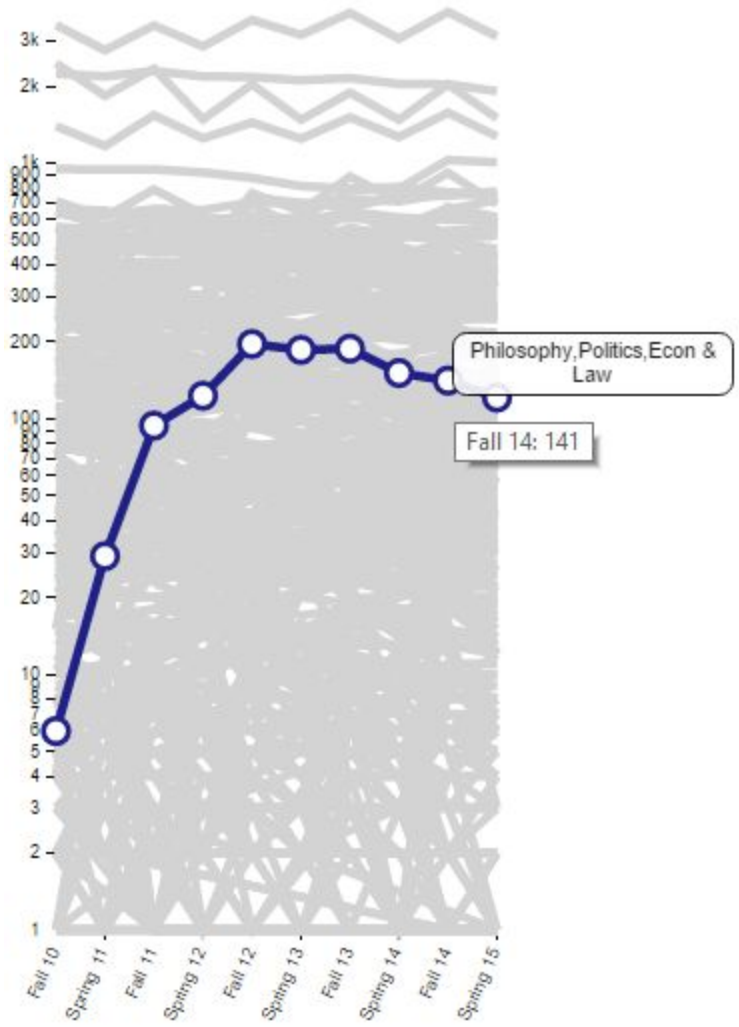

Figure 10. The Philosophy,Politics,Econ & Law major is highlighted in blue. The rest of the majors are less noticeable since they are colored in gray.

After the layout of the graph with all students in each major was optimized to its final state, additional graphs were generated that include only a specific set of students based on their attributes. These splittings include the following categories: gender, ethnicity, and residency status. From the nested dataset of all students for a selected major, we can extract a subset of those students with specific attributes by filtering the set once more. This gives us the dataset that is used for each specialized line graph.

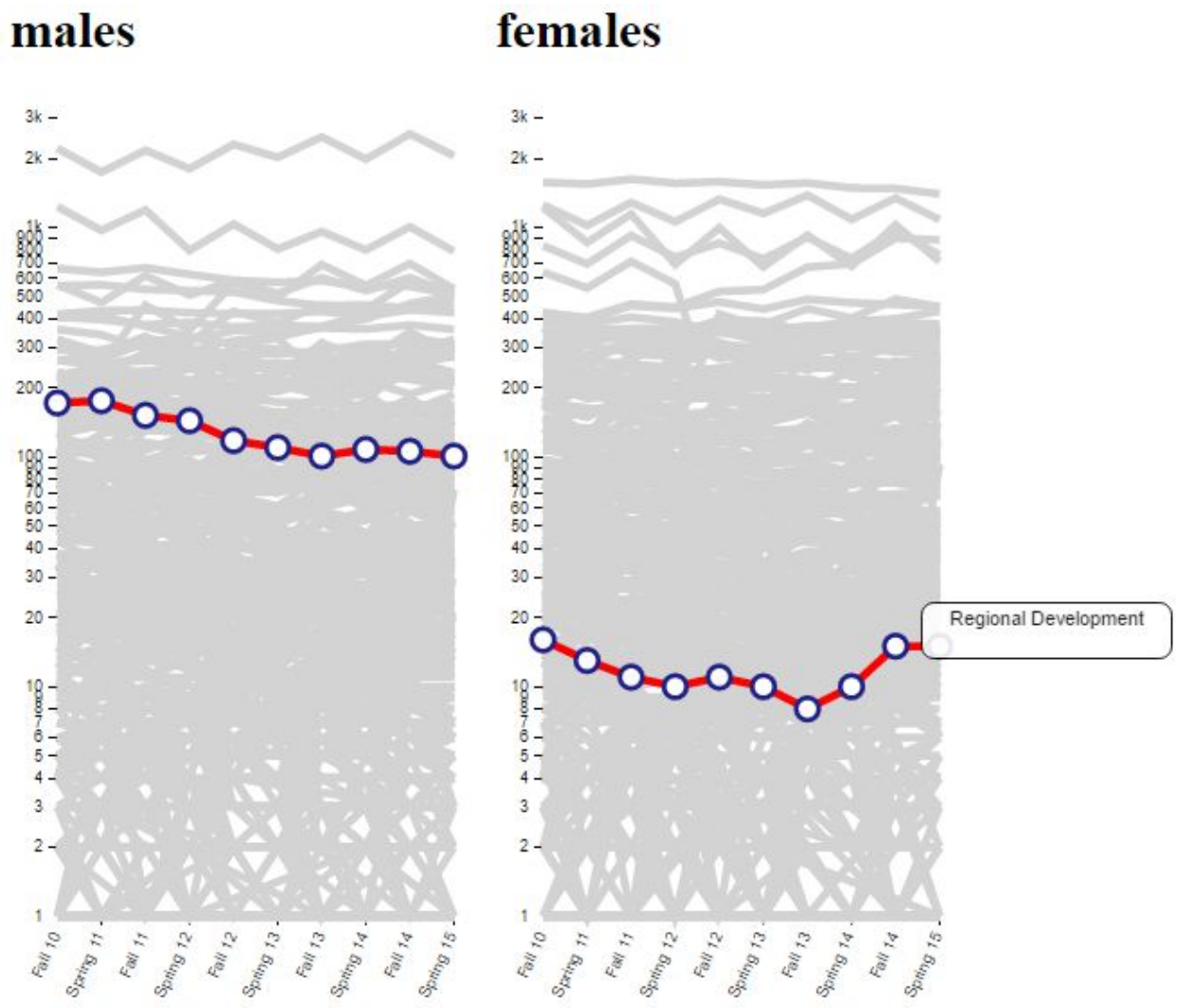

Figure 11. The major “Regional Development” is highlighted in the male and female line graphs.

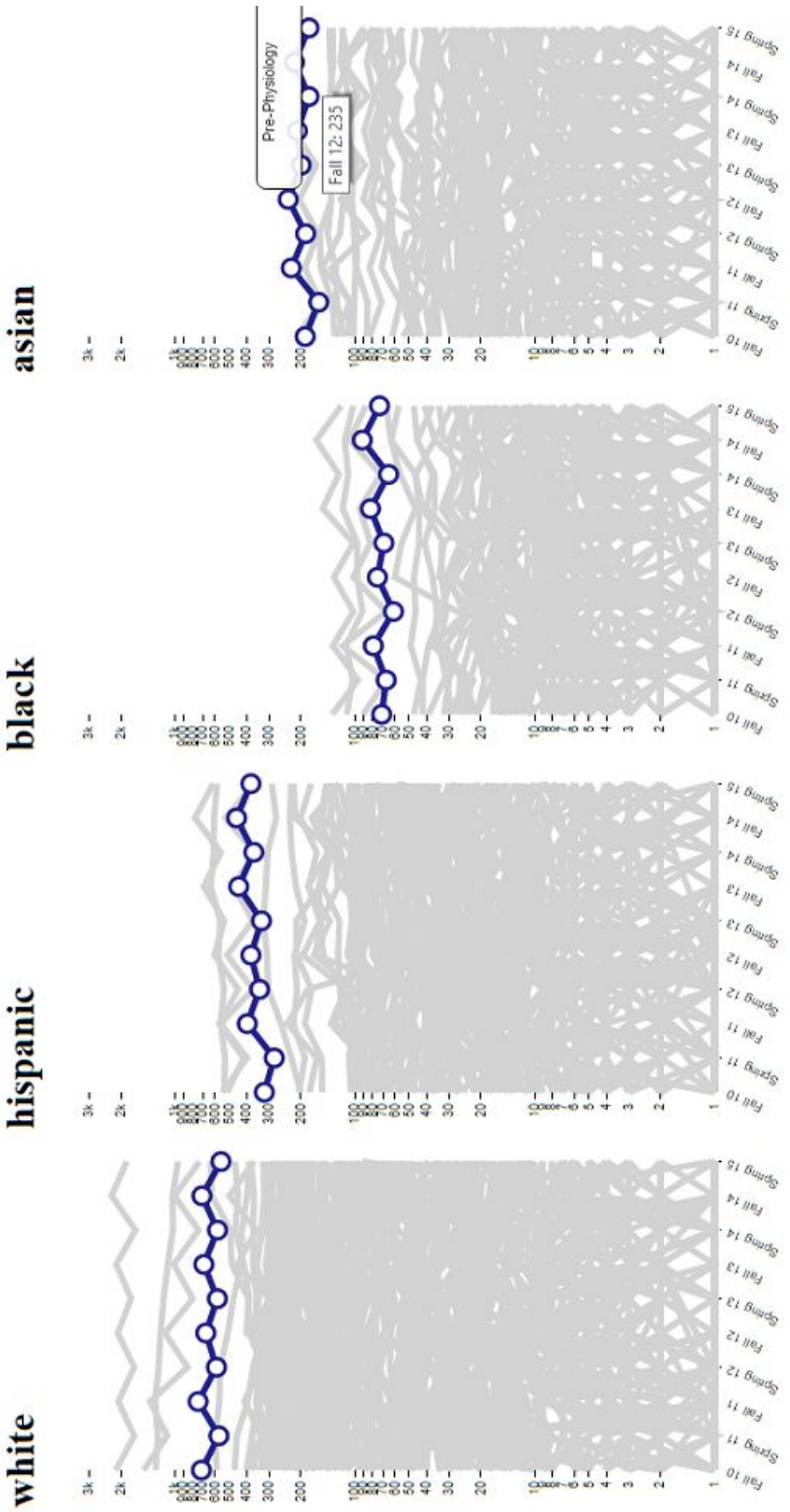

Figure 12. Ethnicity splitting with Pre-Physiology highlighted.

In addition to these splittings, there was one specialized graph created to better understand gender ratios among majors. Titled "gender_diff", the values at each vertex is $log(number\ of\ males) - log(number\ of\ females)$. The values within the equation are synonymous with the values found in the male and female line graphs.

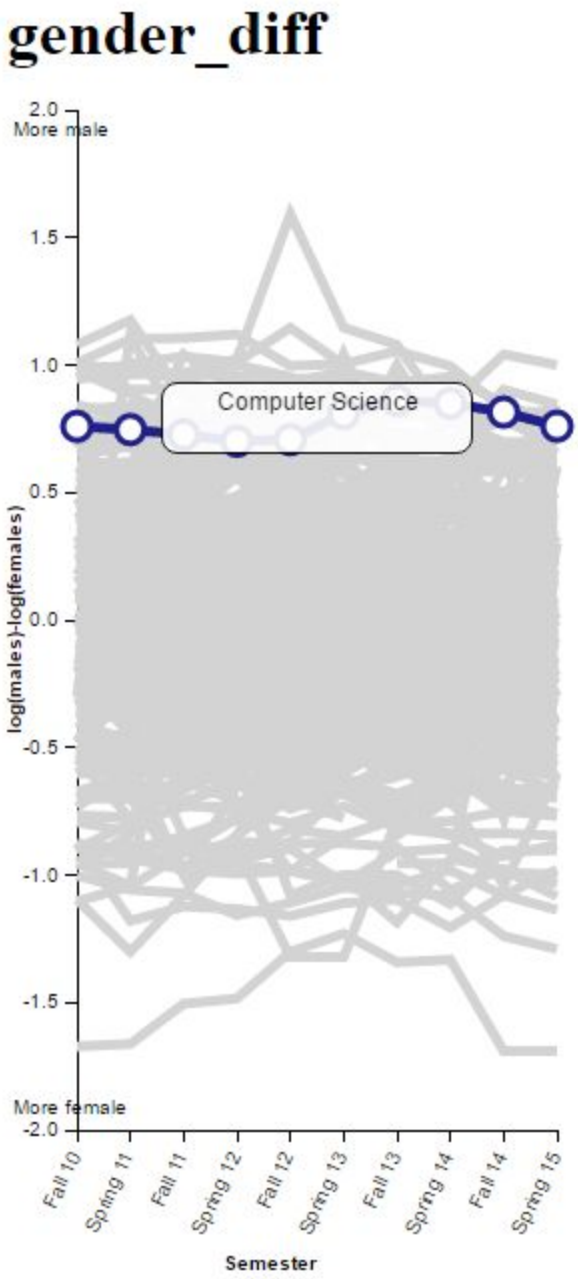

Figure 13. Line graph showing the gender ratios for each major.

One technicality in building this graph was if there were 0 males or females. As by the equation, we would take the log of this value, $log(0)$, but it is undefined. To remedy this, we added 1 to

the number of students for each calculation so that the equation is actually $log(number\ of\ males + 1) - log(number\ of\ females + 1)$. Adding 1 to the value within a log does not change the original calculation to the point where it misguides the information we are trying to convey. Since each calculation in this graph is based off of this equation, the relativity of the line values will remain the same. Applying this solution gives users a clearer picture of the gender ratios across majors.

Tree plot

For the Spring 2016 semester, the focus was to build a visualization that would go back to the original question of how people progress through majors based on the majors they begin with. Charles Joseph Minard illustrated a chart that detailed Napoleon's retreat from Moscow; this chart serves as the primary inspiration for the tree plot we were to build [5].

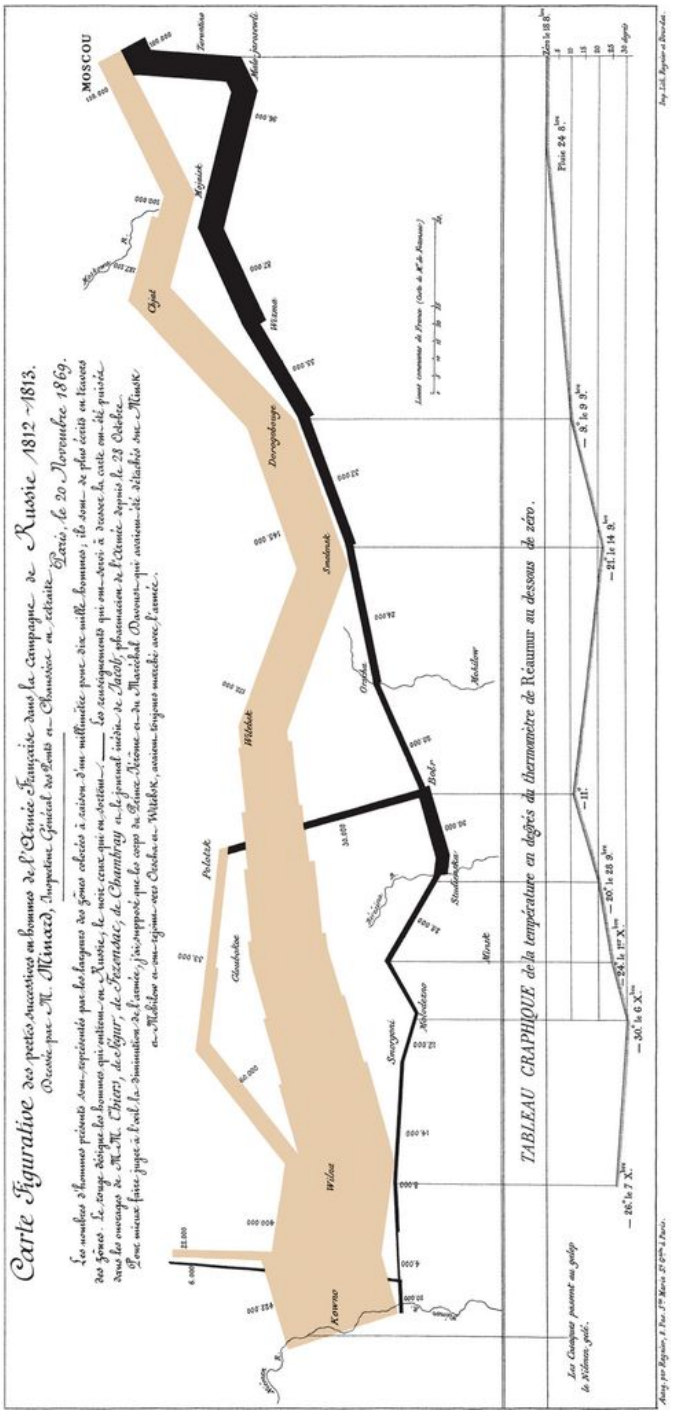

Figure 14. Charles Minard's map of Napoleon's march [5]. This chart conveys how the initial number of soldiers gradually decreased throughout the march to Moscow.

The idea is to begin with an initial subset of students beginning in a specific major and semester. This is the root of the tree. The children of this node represent the students from the root in a different semester. Every node that is a descendent has a subset of students from its

parent and subsequently the root. Because students can possibly be declared a different major from the previous semester, the root can have multiple children since each node represents a different major. Of course, not all students from the parent node will carryover to one of the children nodes because some may have dropped out or graduated. Each level of the tree represents a different semester, and nodes on the same level represent the same semester. The maximum number of levels a tree can have is 10 since the dataset only spans 10 semesters long. The point at which a tree ends means there are no students to report in the next semester.

Formatting the data

A fair amount of data formatting was done to enable the building of a tree. The data structure that would make most sense to model a tree diagram is a tree data structure. To isolate the subset of students given the major and the semester, we began by nesting the data by major then by semester. Using this, I do binary search for the selected major since the key in the dataset represents the major since the majors are sorted alphabetically. Once we find the major, we get the data associated with that major, which is an array of all the semesters. Each semester contains an array of all the students in that major and semester. Since we're getting our subset of students, initially, based on major and semester, the associated semester is binary searched and returned. This was achieved using JavaScript, since we wanted to take advantage of d3's nesting and manipulation of data.

Now that we have the student ID's of the students we're interested in, we need to get information about the subsequent semesters for each student to examine the rest of their academic career. To obtain this information, we create a new nested dataset, this time nested by student ID number and, again do binary search to find each student.

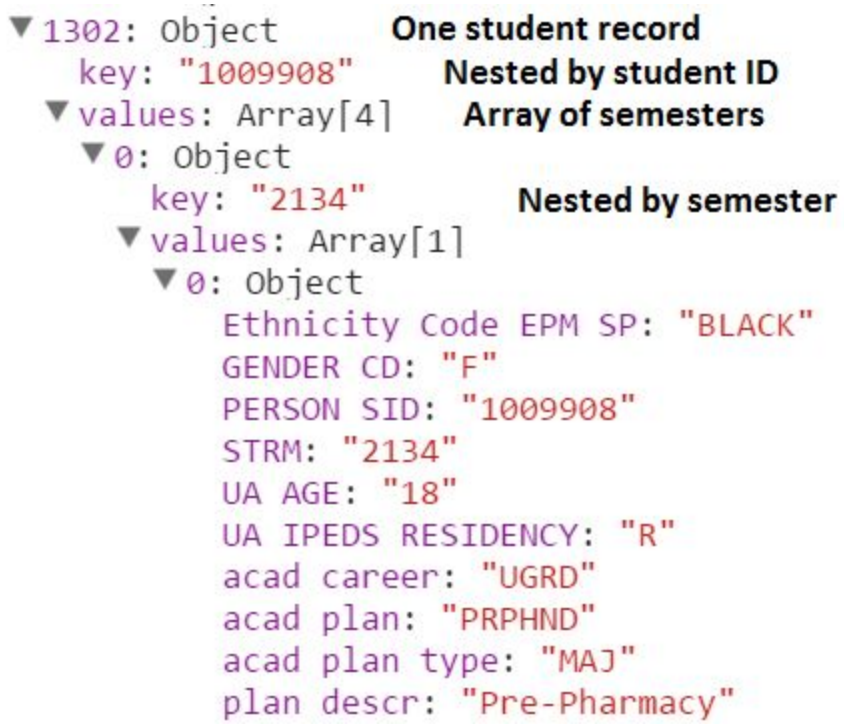

Figure 15. An example of the data nested by student ID then by semester.

From there, we push the student objects that we are interested into a different array. At this point, we have an array of students that are ones whose academic careers we want to examine starting with the semester specified from the beginning. Now we examine each student record, looking at only the specified semester and onwards and build a dollar-sign ($) separated major list. Since we start with a particular major, all students begin with the same major and then branch to others while others stay. The student ID and the dollar-sign separated major list are pushed to an associative array.

```
773291: "Pre-Physiology$Physiology"
773398: "Pre-Physiology$Pre-Physiology"
773543: "Pre-Physiology$Pre-Physiology"
774485: "Pre-Physiology$Pre-Physiology"
775226: "Pre-Physiology$Pre-Physiology"
775404: "Pre-Physiology$Physiology"
775868: "Pre-Physiology$Pre-Physiology"
777329: "Pre-Physiology$Pre-Physiology"
782986: "Pre-Physiology$Molecular & Cellular Biology"
784426: "Pre-Physiology"
784504: "Pre-Physiology$Physiology"
787222: "Pre-Physiology$Pre-Physiology"
787326: "Pre-Physiology"
787826: "Pre-Physiology$Pre-Physiology"
791131: "Pre-Physiology$Pre-Physiology"
792319: "Pre-Physiology"
793139: "Pre-Physiology$Nutritional Sciences"
794381: "Pre-Physiology$Pre-Physiology"
794425: "Pre-Physiology$Physiology"
794606: "Pre-Physiology$Pre-Physiology"
```

Figure 16. Associative array containing student ID's as the key and a dollar sign separated major list. This example's selection Pre-Physiology 2144 (Fall 2014). Since there are only 10 semesters in this dataset starting with Fall 2010 and ending with Spring 2015, the displayed majors end at Spring 2015, or earlier if there are no semesters that the student displayed beyond Fall 2014.

```
103104: "Pre-Physiology$Pre-Physiology$Pre-Physiology"
103123: "Pre-Physiology$No Major Selected Ltr Art Sci$Psychology$Psychology$Psychology$Psychology$Psychology$Psychology"
103170: "Pre-Physiology$Physiology$Physiology$Physiology"
103986: "Pre-Physiology$Pre-Physiology$Physiology$Physiology"
105296: "Pre-Physiology$Neuroscience$Neuroscience$Neuroscience"
105376: "Pre-Physiology$Pre-Physiology$Pre-Physiology$Pre-Physiology$Pre-Physiology$Family Studies & Human Dev"
106174: "Pre-Physiology$Physiology$Physiology$Physiology$Physiology"
106358: "Pre-Physiology$Physiology"
107286: "Pre-Physiology$Microbiology$Microbiology$Microbiology"
108039: "Pre-Physiology$Pre-Physiology"
109134: "Pre-Physiology$Pre-Physiology$Psychology$Psychology$Psychology"
```

Figure 17. Associative array representing the same idea as Figure 16 but starting from Fall 2010.

Next we build a tree based on the associative array. To begin this, we had to define a Node object and come up with variables that are relevant such as an array of children, the level, the count of students, etc. We build the tree recursively starting with the root, which contains the subset of students for the specified major and semester. The result is a root that represents the initial set of students. One level below that, its children, is the subset of students for the next semester. The children of a node are where students ended the next semester.

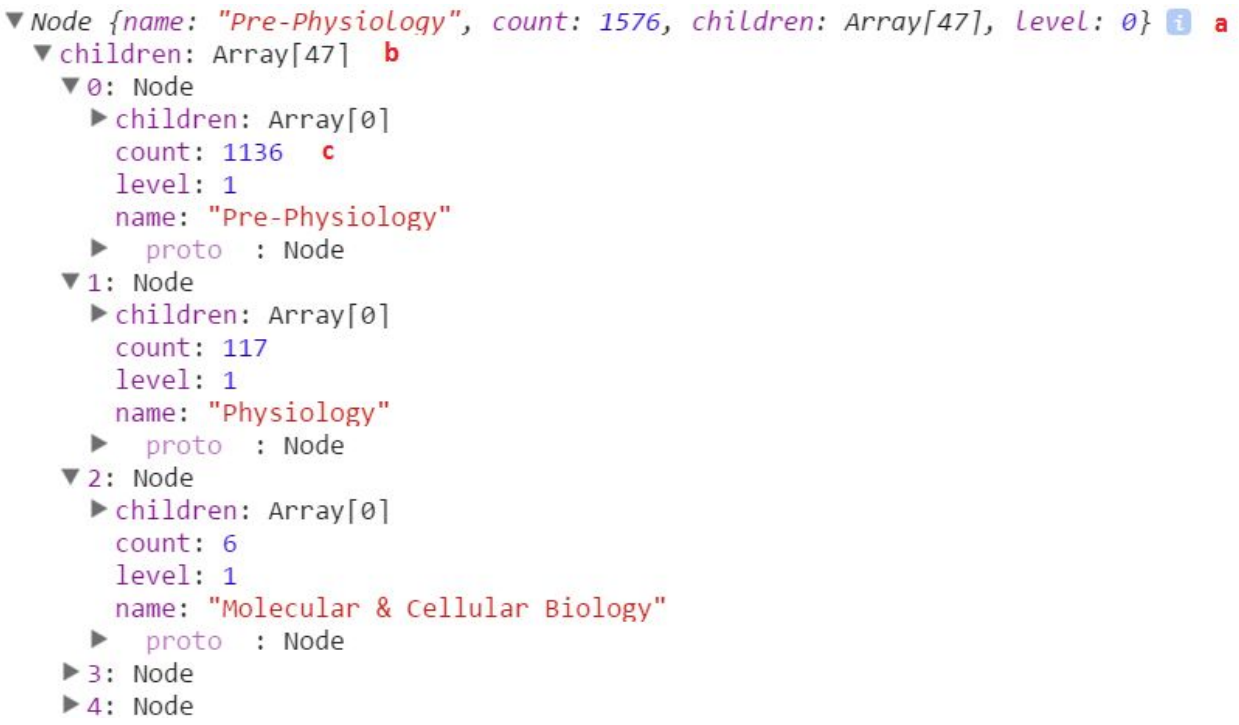

Figure 18. An example of a tree for 2 semesters. (a) The root of the tree includes all the students with Pre-Physiology as their major (1576) in the Fall 2014 semester. (b) The root has 47 children meaning there were 47 different majors, including the root major, students from the initial set moved to the next semester (Spring 2015). (c) 1136 students stayed in the Pre-Physiology major, while 117 moved to Physiology, 6 moved to Molecular & Cellular Biology. These are only 3/47 majors students moved to from Pre-Physiology in Spring 2015.

Building the tree

D3 supplies a built-in tree layout that can be applied to any data set of the correct structure [14]. We decided to use the basic tree layout as a starting point since it includes basic layout features of interest. This includes dynamic node spacing features and different levels. To use d3's built-in tree layout, the data must come in the form of a JSON array. The data structure we created resembles the required data format, so we were directly able to apply the d3 tree layout to the dataset.

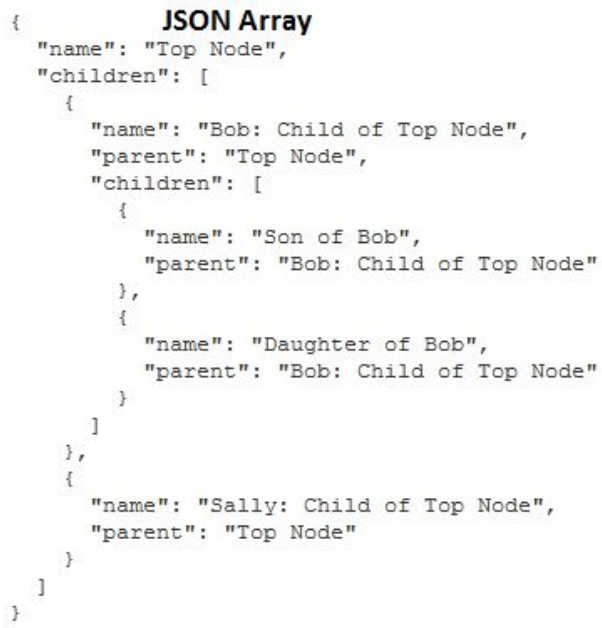

Figure 19. The JSON array specifies the name of the object, children, and parent [13].

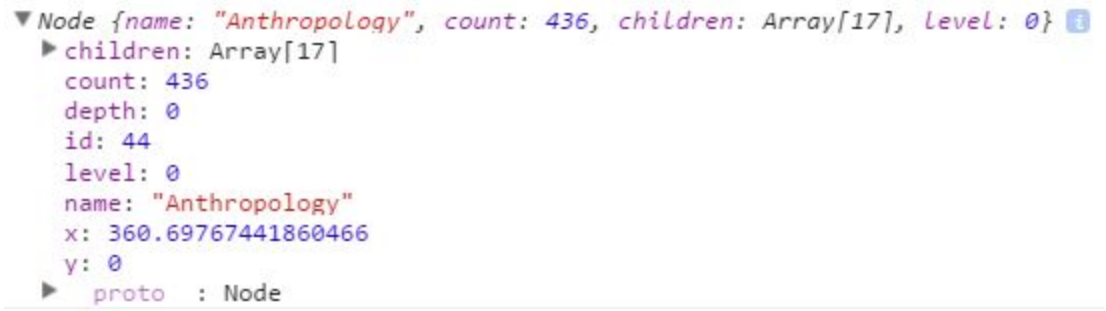

Figure 20. Our data structure specifies a name, children, and other variables.

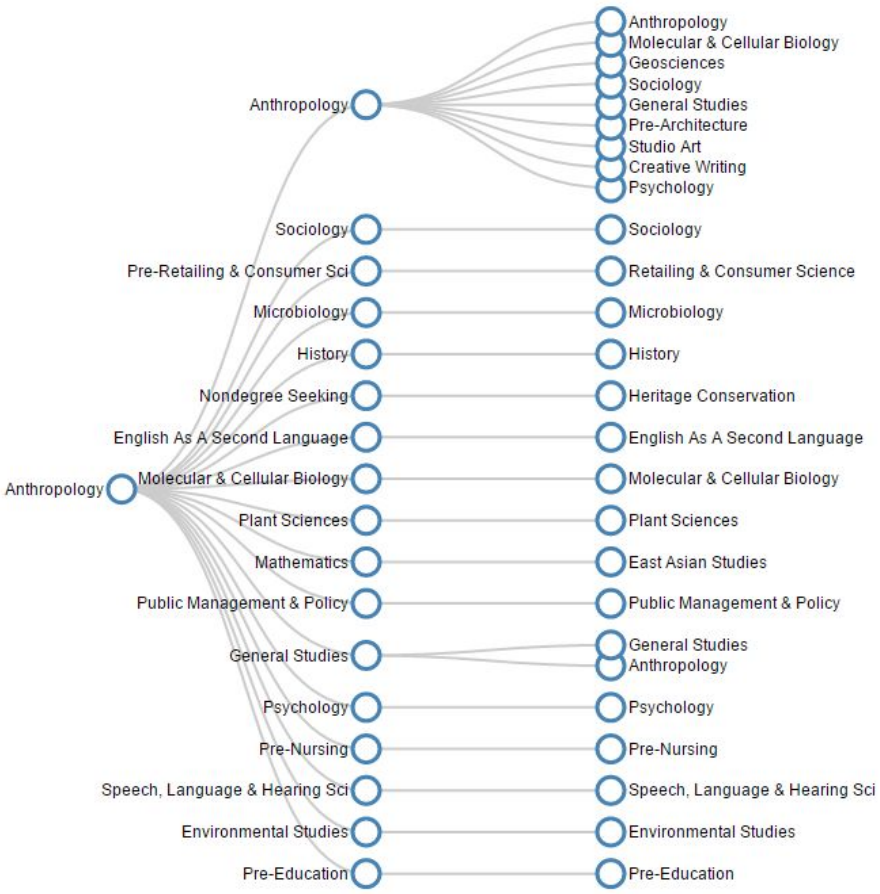

Figure 21. Tree svg of students starting in semester 2141 (Spring 2014). This tree ends with Spring 2015 since this is as far as the data we have. The 3 levels (columns) of nodes represents 3 separate semesters: Spring 2014, Fall 2014, Spring 2015.

Adjusting node spacing

As a way to eventually move towards having the size of nodes being proportional to the number of student a node represents, the space around a node must first be adjusted. We decided that we should use the spacing encoded in d3's tree layout to our advantage to create the amount of space we wanted. The number of children nodes directly increases the space of the parent between nodes on the parent's level. With this idea in mind, we decided that for each leaf within the tree, we could add a number of nodes with empty values to the leaf's children that are proportional to the number of student the leaf holds. The leaf node will have more space around it. In turn, the spacing recursively moves up the tree until we reach the root. We refer to these additional empty nodes used for spacing purposes as "ghost" nodes.

We modified the buildTree method so that if a node had no children and it hit the base case, it would add a ghost node.

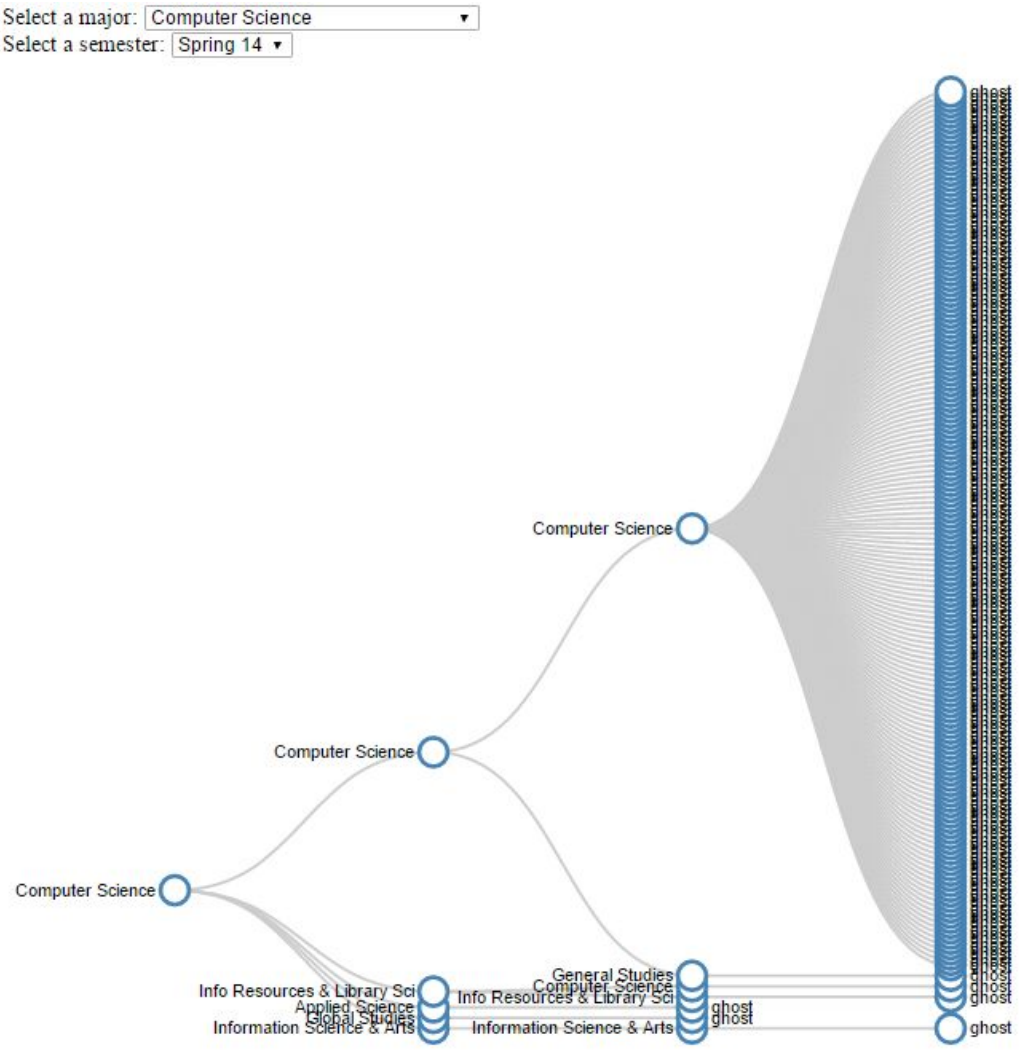

Figure 22. Added ghost nodes to tree.

Figure 23. Made links to ghost nodes hidden visibility. As a result, a big space is left where hidden fields are.

This worked until some nodes that were not leaves were producing ghost nodes. This was because sometimes, depending on the order of students, the root would not have children for a while as students are added since they do not stay beyond a semester. So the root would be a leaf temporarily causing us to add a ghost node. To resolve this, we add a boolean in the Node structure called isLeaf. It is initially set to true for all nodes and set to false as children are added to a node. After we buildTree, we traverse the whole tree, look for the isLeaf nodes, and add ghost nodes corresponding to the count.

Since the spacing looked tight between nodes with less students, we reduced the size of ghost nodes drawn for nodes whose counts were greater than 50. If the count is greater than 50, we divide that value by 5 until the value is less than 50. The result is the number of ghost nodes drawn for that node.

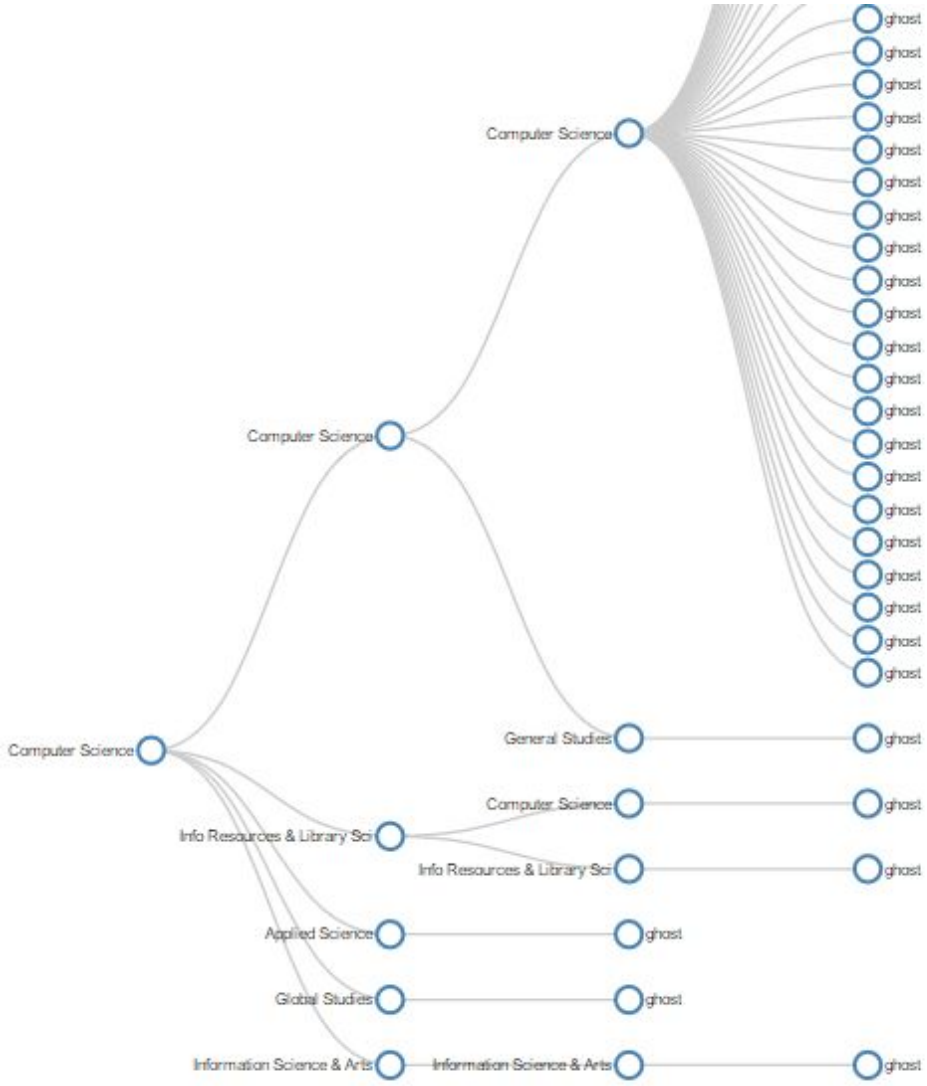

Figure 24. Number of ghost nodes are reduced to take up less room. This gives more space to nodes with less people.

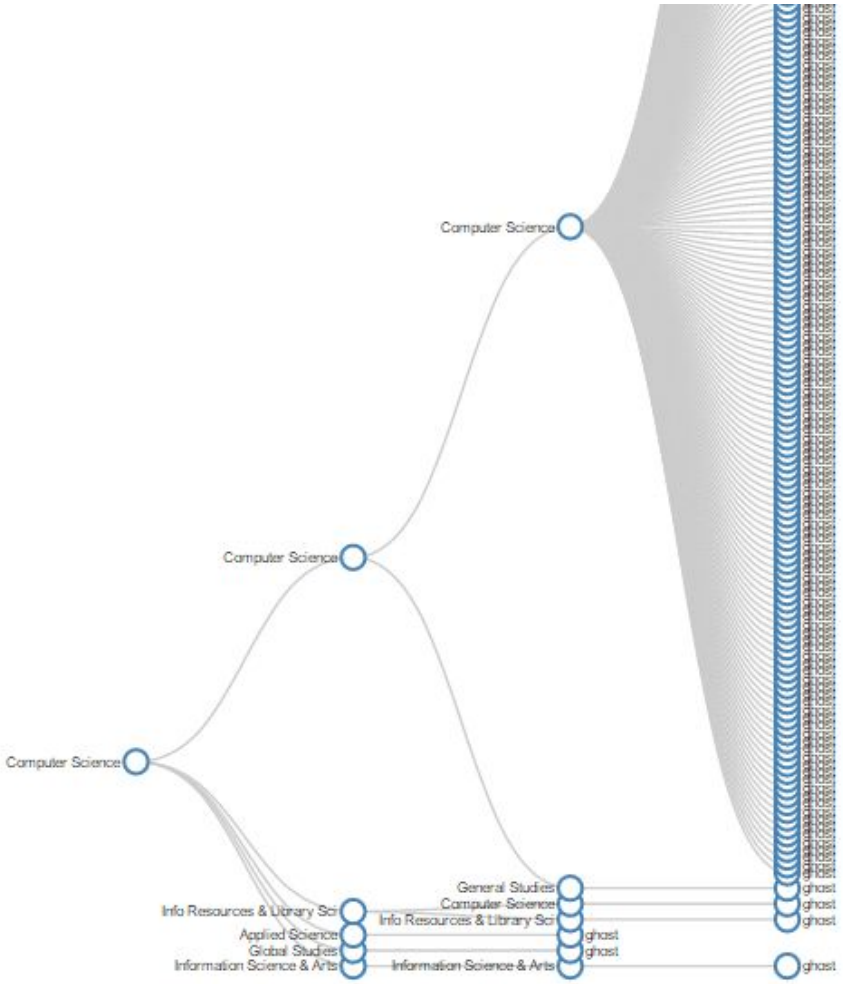

Figure 25. The same graph as Figure 24 but without adjustment of number of ghost nodes. The spacing of the nodes at the bottom look squished.

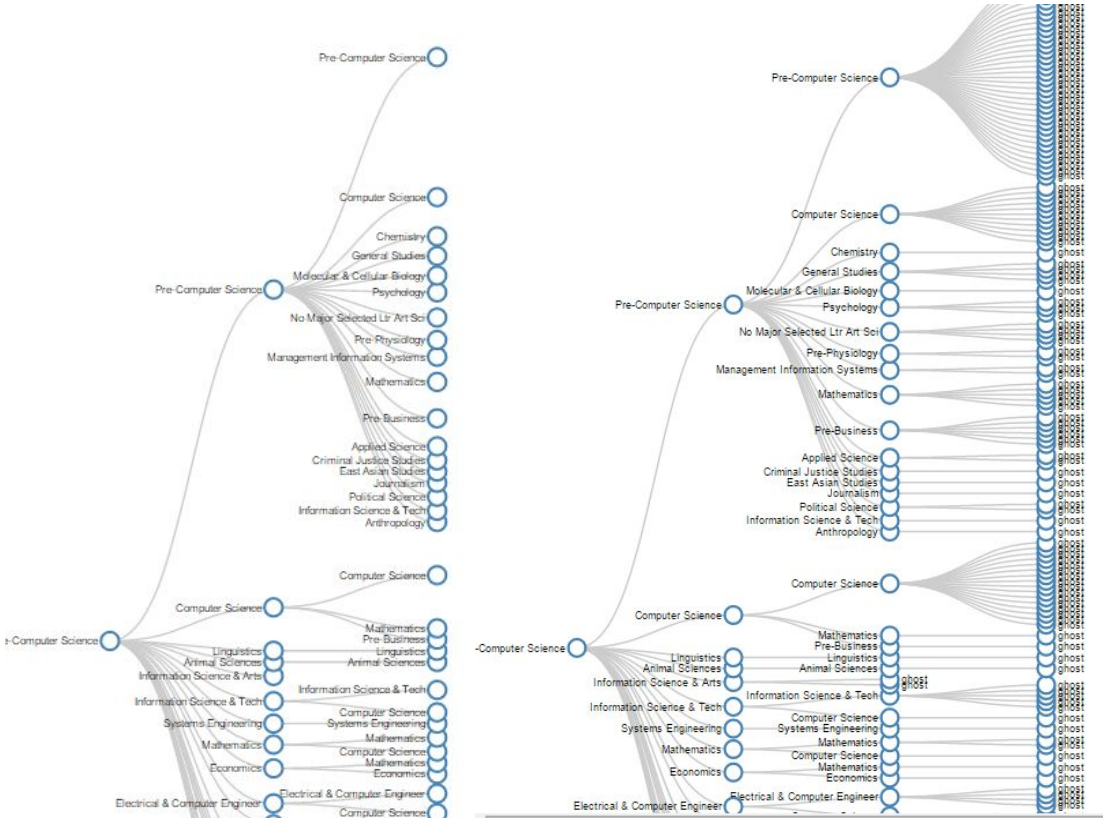

Figure 26 (left). Starting in Spr. ‘14 for Pre-Computer Science. There is more branching and the spaces do correspond to the number of people, which is helpful. Figure 27 (right). This is the same as Figure 26 but with the ghosts shown.

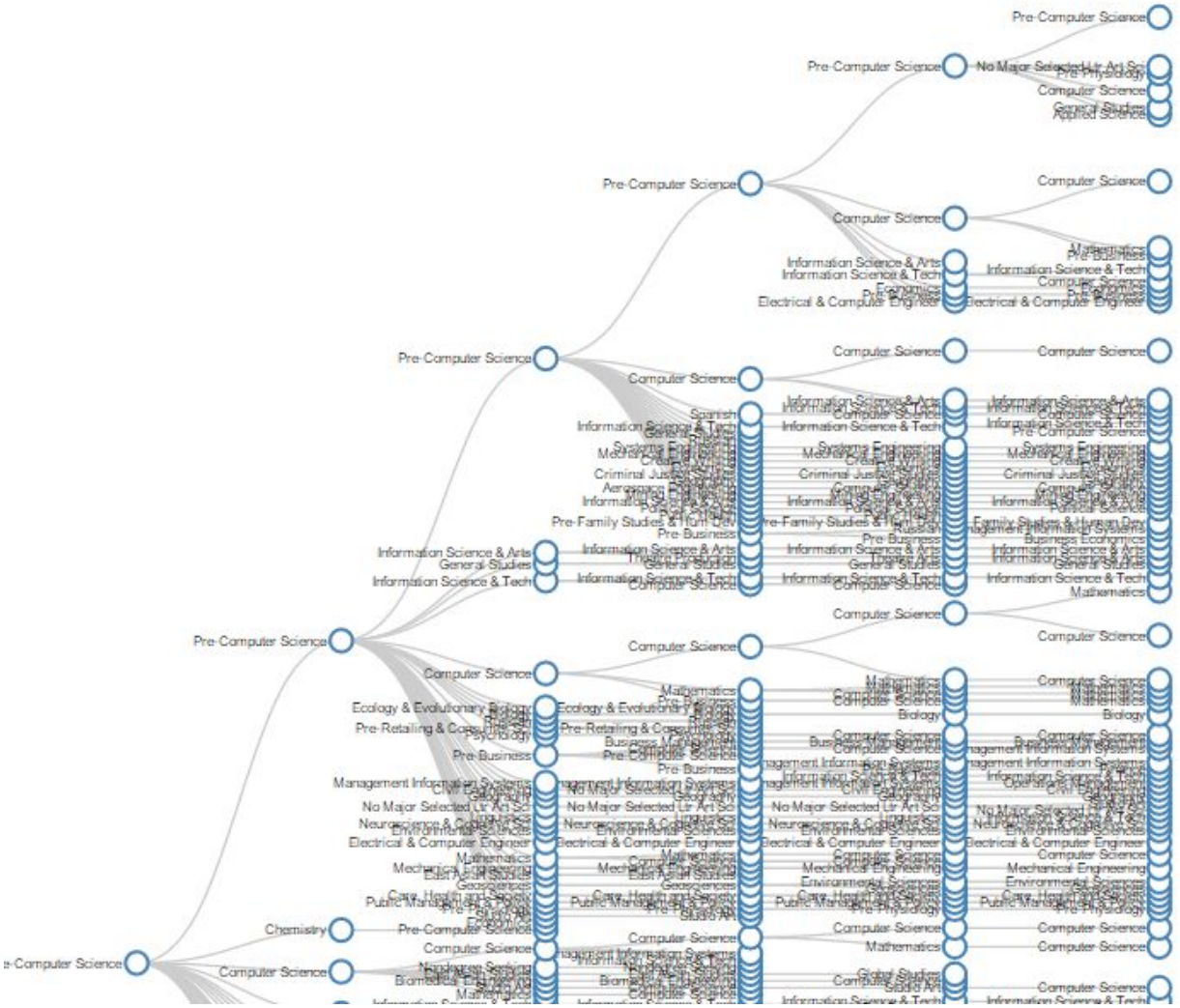

Figure 28. Starting Fall ‘12 for Pre-Computer Science. Starting from an earlier semester quickly gives us a scarier, more cramped graph very easily.

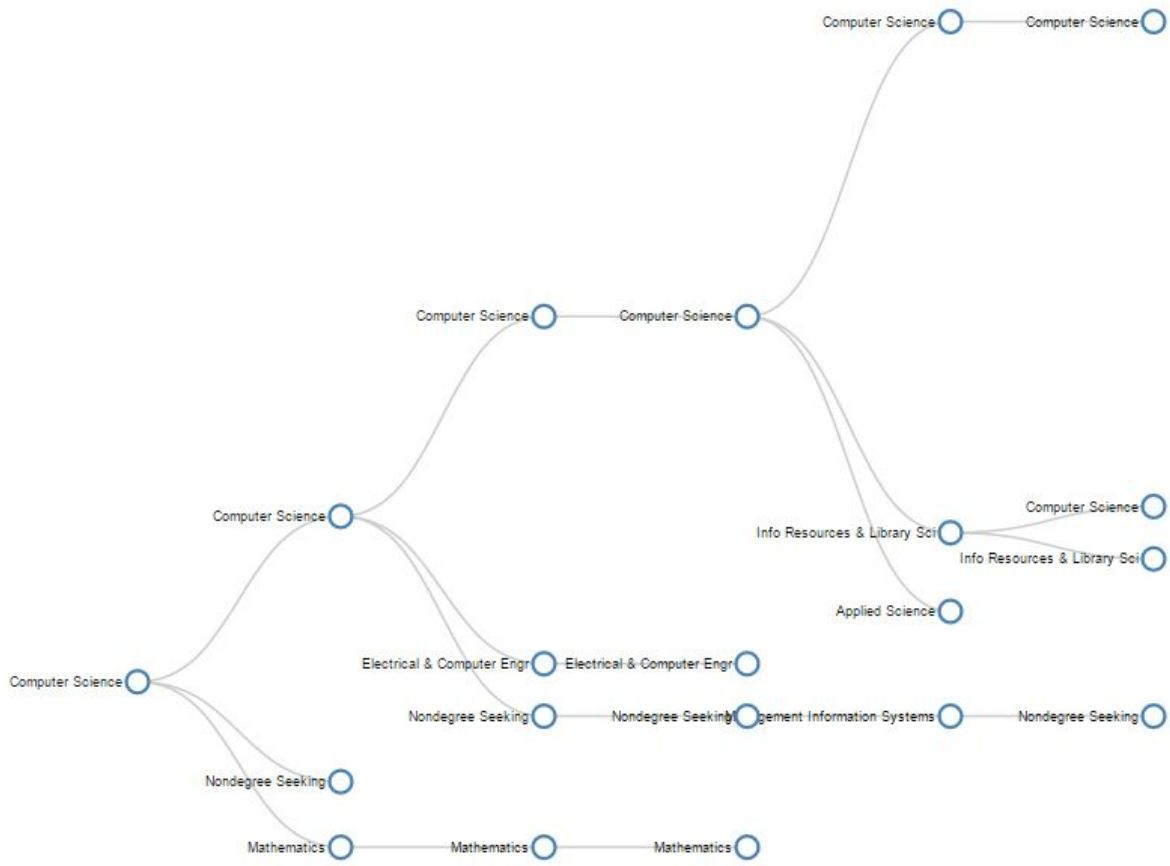

Figure 29. Starting Fall ‘12 for Computer Science. Since there is less movement in Computer Science than Pre-Computer Science, it produces a much neater looking tree.

Adjusting node size

Having the node size relative to the number of students in the subset can greatly improve the quality of the visualization. Since we already having the spacing in place to accommodate a larger node, we need to find out how to adjust the sizing of nodes so that it makes sense. The algorithm for this should be able to be adapted to all nodes in the tree. This requires some calculations of the location of ghost nodes that we made previously since this will be the space we are working with. Here we worked with the x-values of the nodes. Although normally this would be the y-value on a normal graph, we refer to them as x-values because d3’s tree algorithm builds the tree vertically and then rotates it 90 degrees so that the x and y-axes are flipped.

Attempt 1: height = max.x - min.x of children

We tried finding the height of a parent node by taking the child with the max.x-value and subtracting it from the child with the min.x-value, but as we went up the tree, it did not span the heights of the nodes like we were expecting. Instead, it was only the x-value where the svg elements were placed, subtracted from each other.

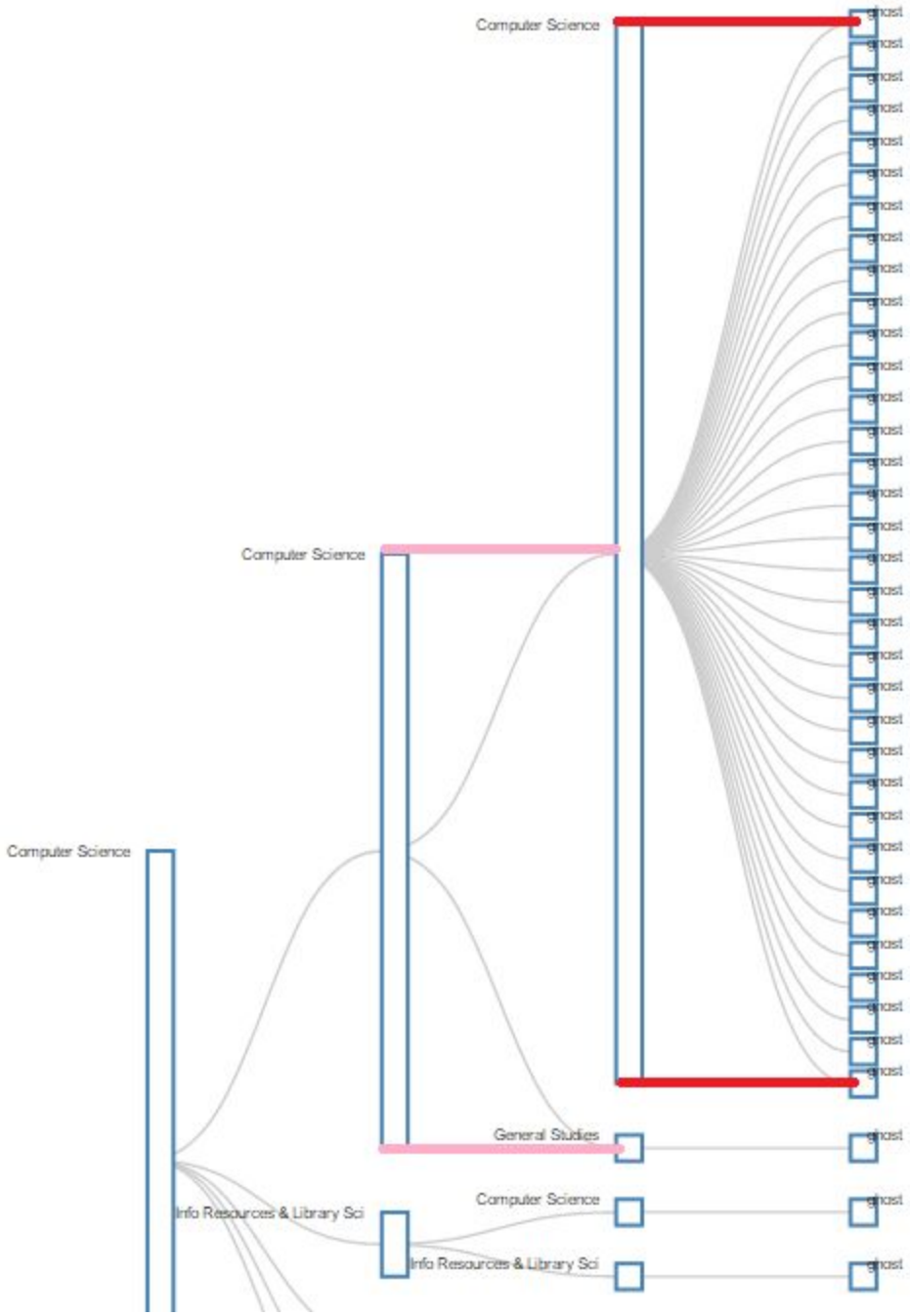

Figure 30. The colored lines are the min and max values of the children for the parent node.

We wanted the top of the svg element of the max and the bottom of the svg element of the minimum to be the height of the parent.

Attempt 2: height = the sum of the heights of all its children

Next we tried setting the height to the sum of the heights of all the parent's children using post-order traversal. This gave an accurate number of height, but did not include the spaces between the nodes. Also since this was processed before the tree was drawn, there was no x-values to go off of since nothing was drawn yet.

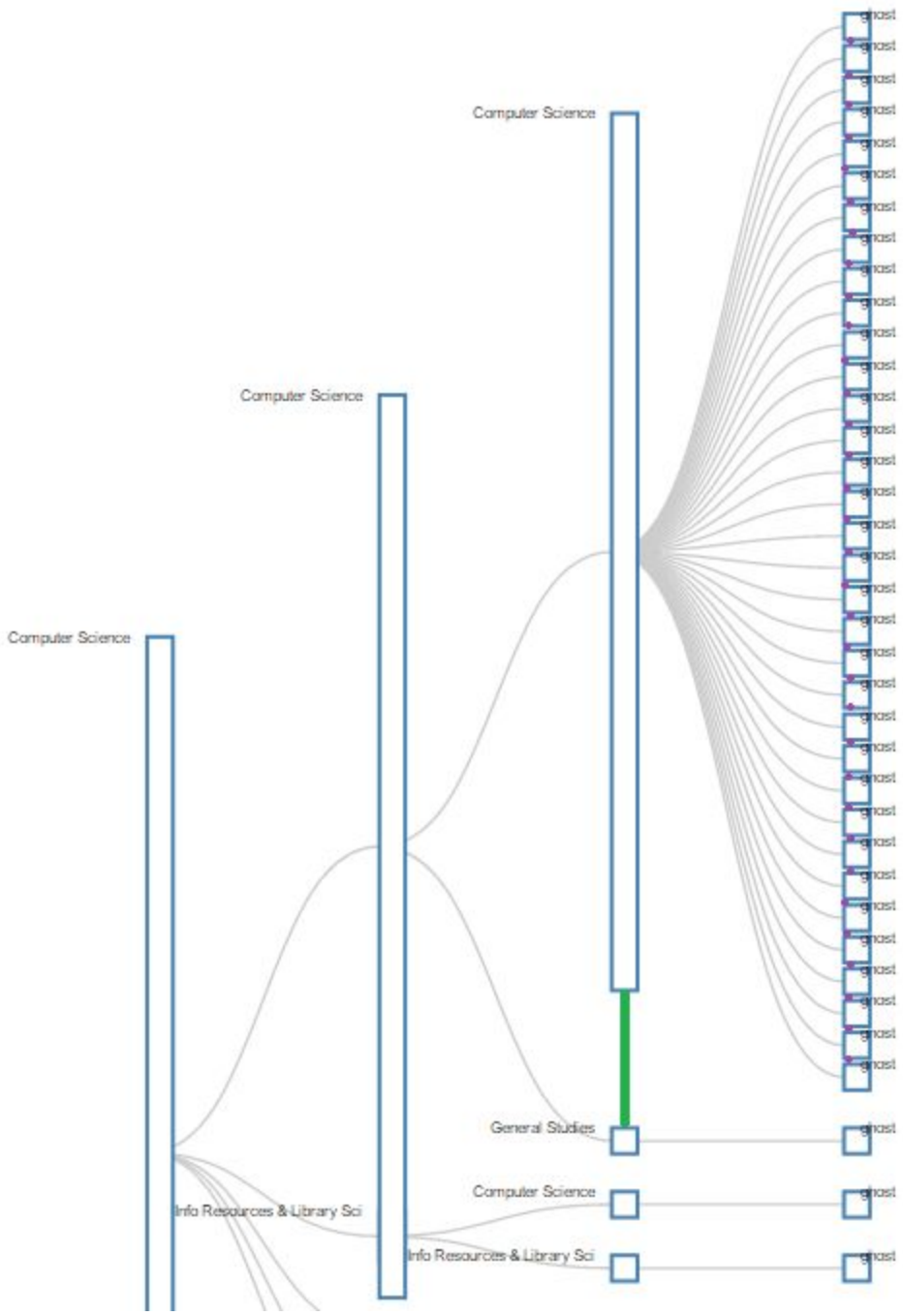

Figure 31. Colored points (green and purple) are height amounts unaccounted for, for each of the parent nodes.

Attempt 3: height = max.x - min.x + (half of the size of the max shape) + (half of the size of the min shape)

In order to be table to use the x-values of the tree diagram, we do the height declaration while drawing the tree. This works since the tree is drawn from the leaves to the root. This next attempt takes into account the space between the children nodes. This is done by taking the top of the max child which is calculated using:

top of max child = max.x + max.height/2

Likewise, the bottom of min is found using:

bottom of min child = min.x + min.height/2

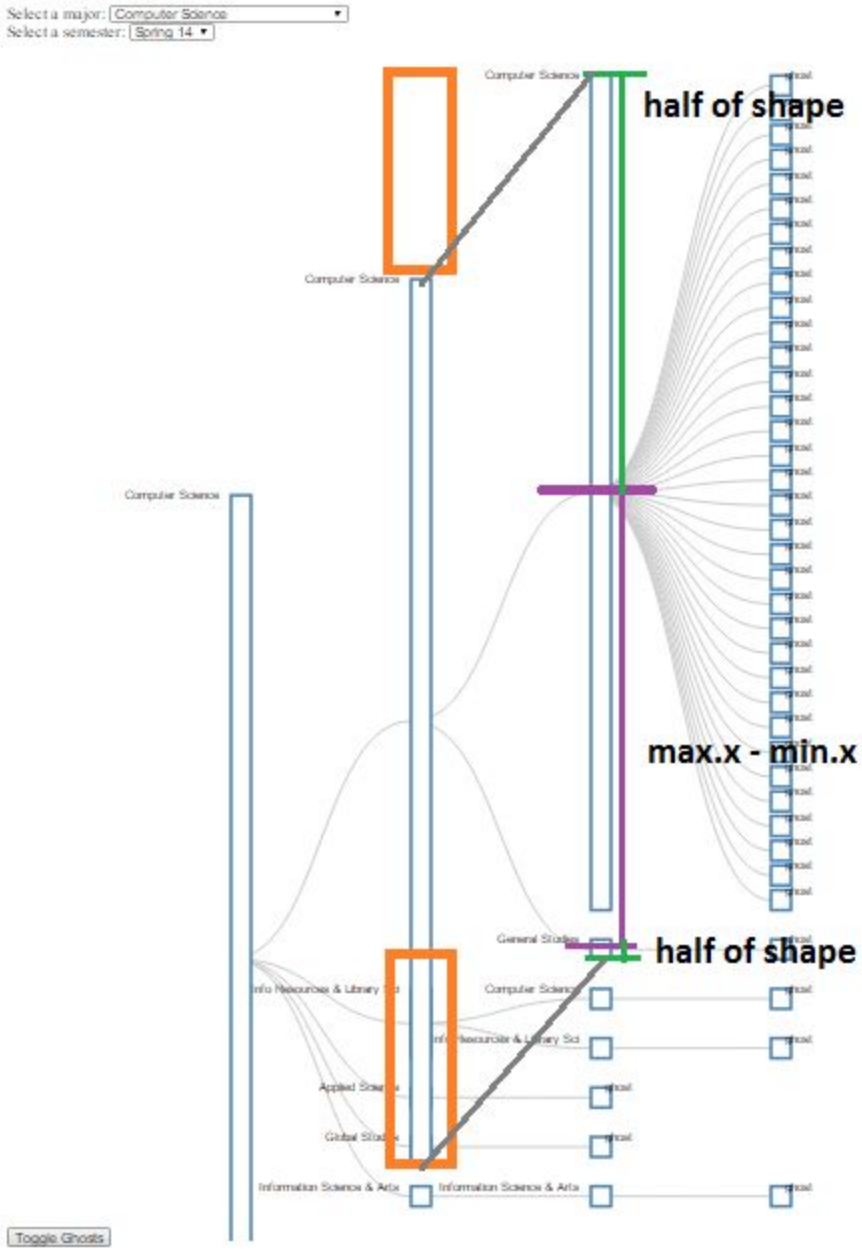

Figure 32. The height of the node with orange boxes is calculated by first finding the purple distance. Next we take ½ of the height of those shapes. The sum of all these gives us the height of the parent node. However, the orange boxes show how the bottom portion covers up other nodes and should be shifted to the top orange box.

Attempt 4: Using Attempt 3 to find the height and shifting the nodes so that the top of the parent is the same as the top of the highest child

Before, the y-position was found by taking the element's x-position and subtracting the element's node height by 2 so that the link would fall in the middle of the node. We had to modify it again so that instead of having the link fall in the middle of the node, the y-position of the parent node would match the y-position of the min.x element. This was found using:

y-position = highest child's y-position + (height of highest child/2)

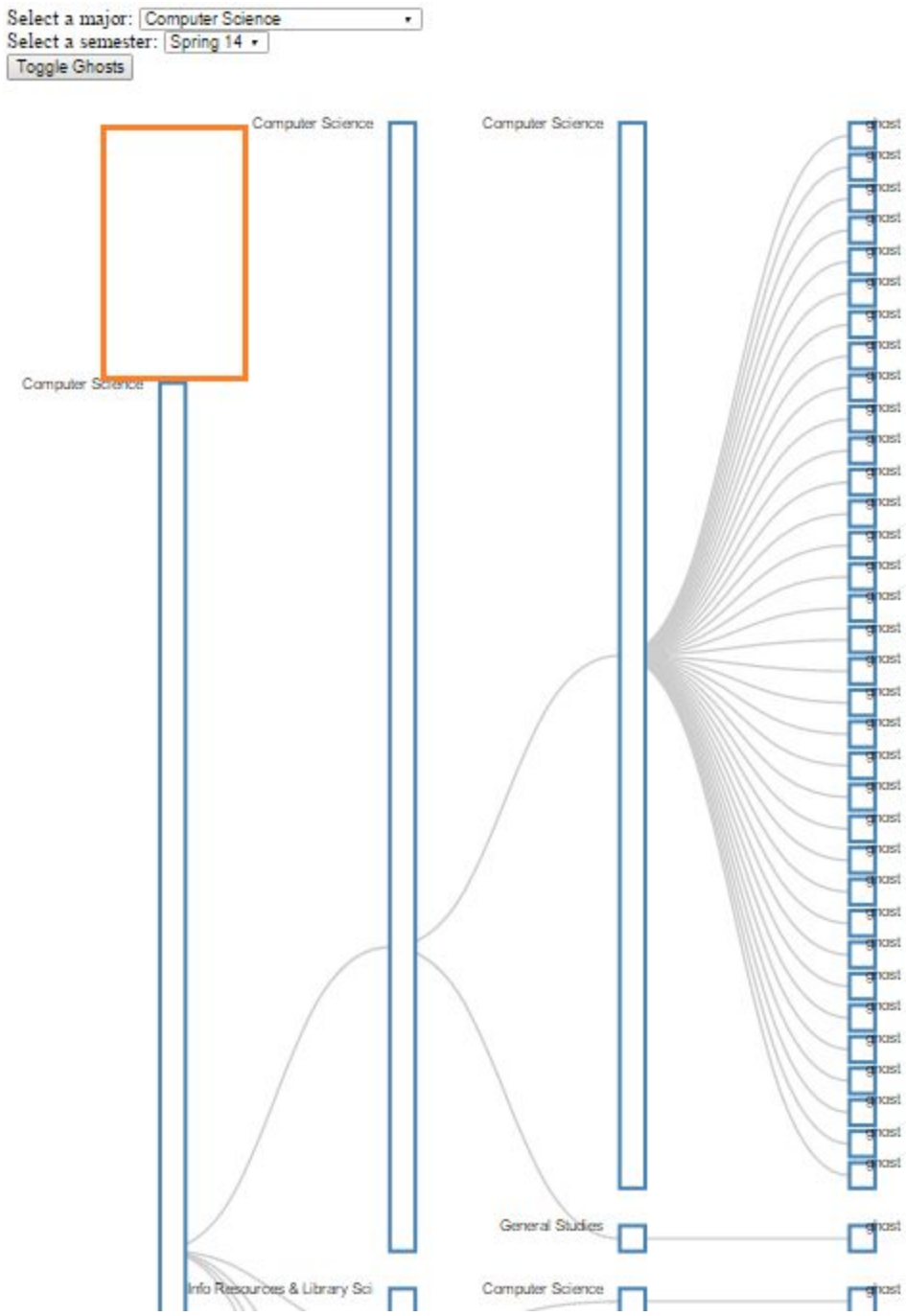

Figure 33. The middle two Computer Science nodes are aligned at the top. This does the job until a parent that uses a child's old y-position to find its new position is encountered as seen by the orange box.

This is pretty good until a parent attempts to use the old, unshifted y-position of a child to find its new y-position.

Attempt 5: Amending the height and x(y) value of a node

We changed the way in which we approached this by pre-processing the heights and x-values before the tree was beginning to draw. The same way the y-values of each node was changed before the drawing began, we changed the x-values. This was done by using post-order traversal to traverse the tree, finding the min and max x-values, and setting the height field of each node based on their children. The key was to ignore the position of the links and modify the x-values rather than trying to individually translate and calculate the heights of each node during the construction of the tree.

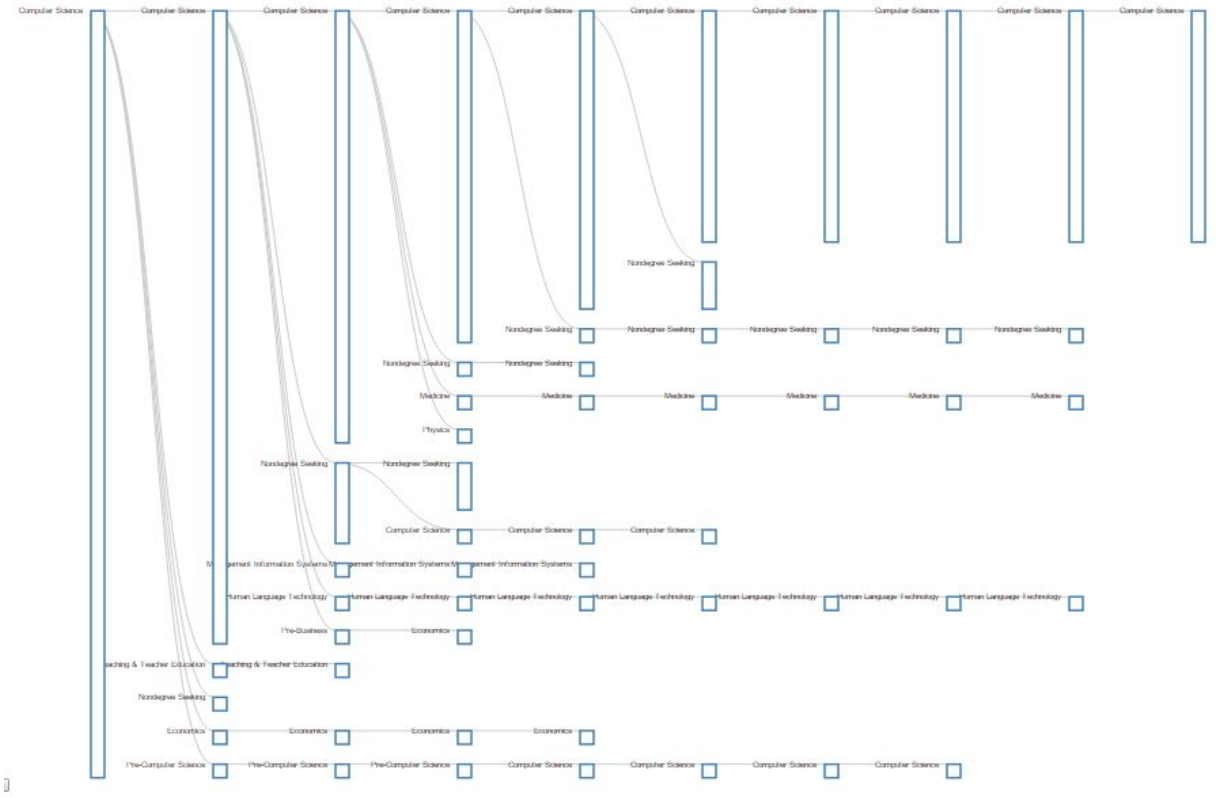

Figure 34. Computer Science Fall ‘10 is shown. Heights correspond to the heights of their children. Toggling ghosts will make this more obvious, but the highest child decides the parent’s x-position and the difference of the highest child and lowest child plus the height of the lowest child gives us the height of the parent.

The links connect each node based on these x-values. This is why they connect the top of nodes rather than the middle as before. The x-value is also where the top of each node is drawn.

Adding Ghost Nodes to Max Depth

To fill in the tree with ghosts nodes that go to the max depth of the tree, we have to find max depth. We did this by taking advantage of the depth field added when d3 adds attributes to the tree when you use d3.tree.layout. Another way we could do this is by traversing the tree and keeping track of the depth.

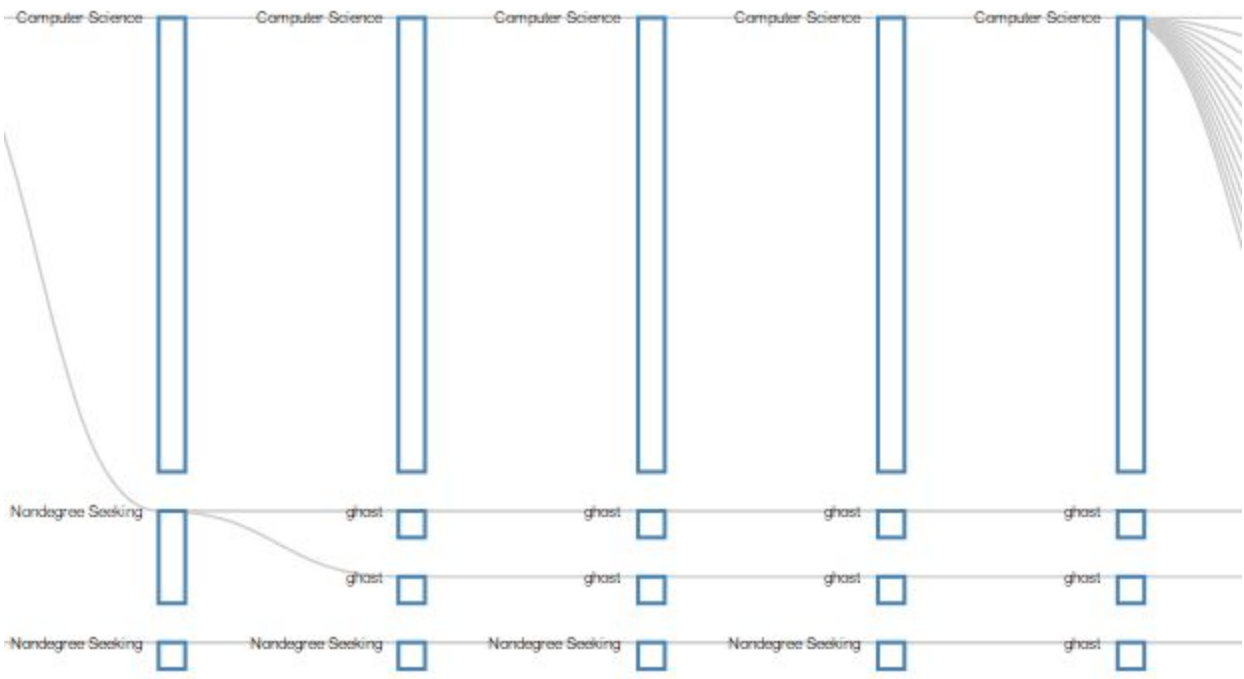

Figure 35. Ghost nodes are added all the way to max depth, which in this case is 10.

Ghosts nodes are added all the way to the end because nodes were being overlapped by others before since the tree layout only gives you enough room for nodes in the current level.

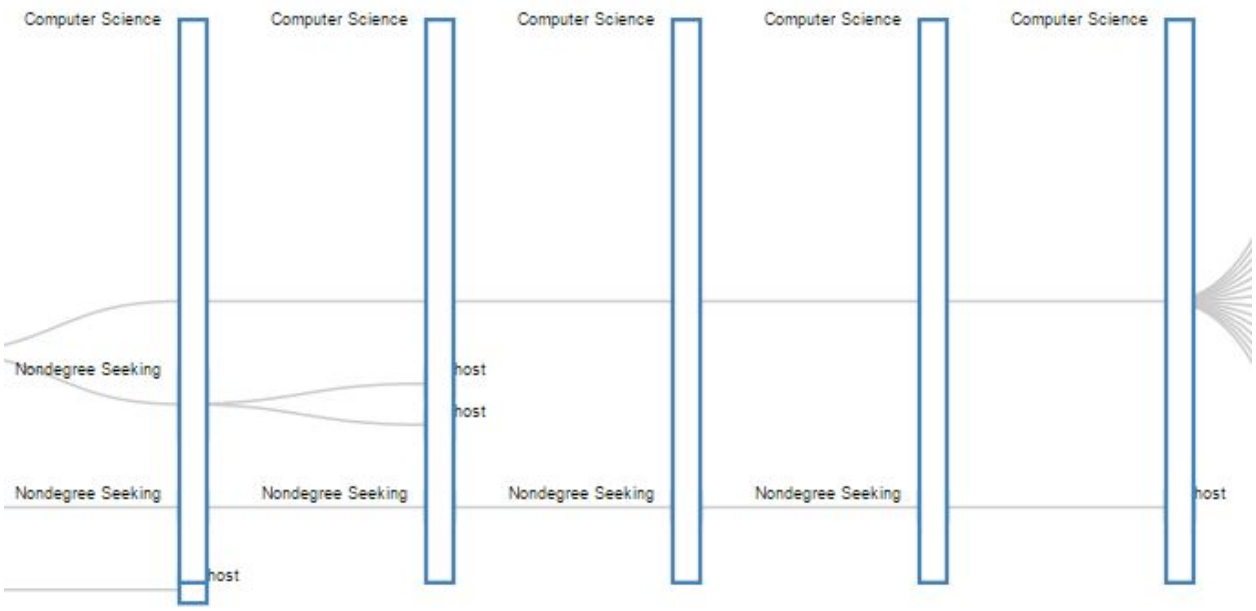

Figure 36. In the same graph before ghost nodes are drawn to the ends, we can see that the Nondegree Seeking nodes get covered by the Computer Science nodes.

Adjusting Height of Tree

On many occasions, a tree will be too big to fit within a fixed-sized svg. This causes the trees to look cramped. We decided that the height of an svg element should be relative to the number of students for each set of data. To do this, we needed to find the number of ghost nodes at the bottommost level and multiply it by how much space we wanted a ghost node to occupy. We had to come up with some logic to get the number of ghosts at the bottommost level and not those at higher levels that were added from before. From there, the height of the svg was calculated by multiplying a constant by the number of ghosts at the bottommost level.

Changing positions of links/text

To change the position of the text, when the text field was being added, we adjusted the y-position of the text to be ½ of the height of the node's rectangle height which we already store in the Node object. This shifted the y-position of the text by that amount since it is contained within the g element. To make it so that the position of the links fall within the middle of each node, we re-processed the x-values for each node by adding half of the height of the node to the current y-position. So this shifted the values down by half of the height of the rectangle.

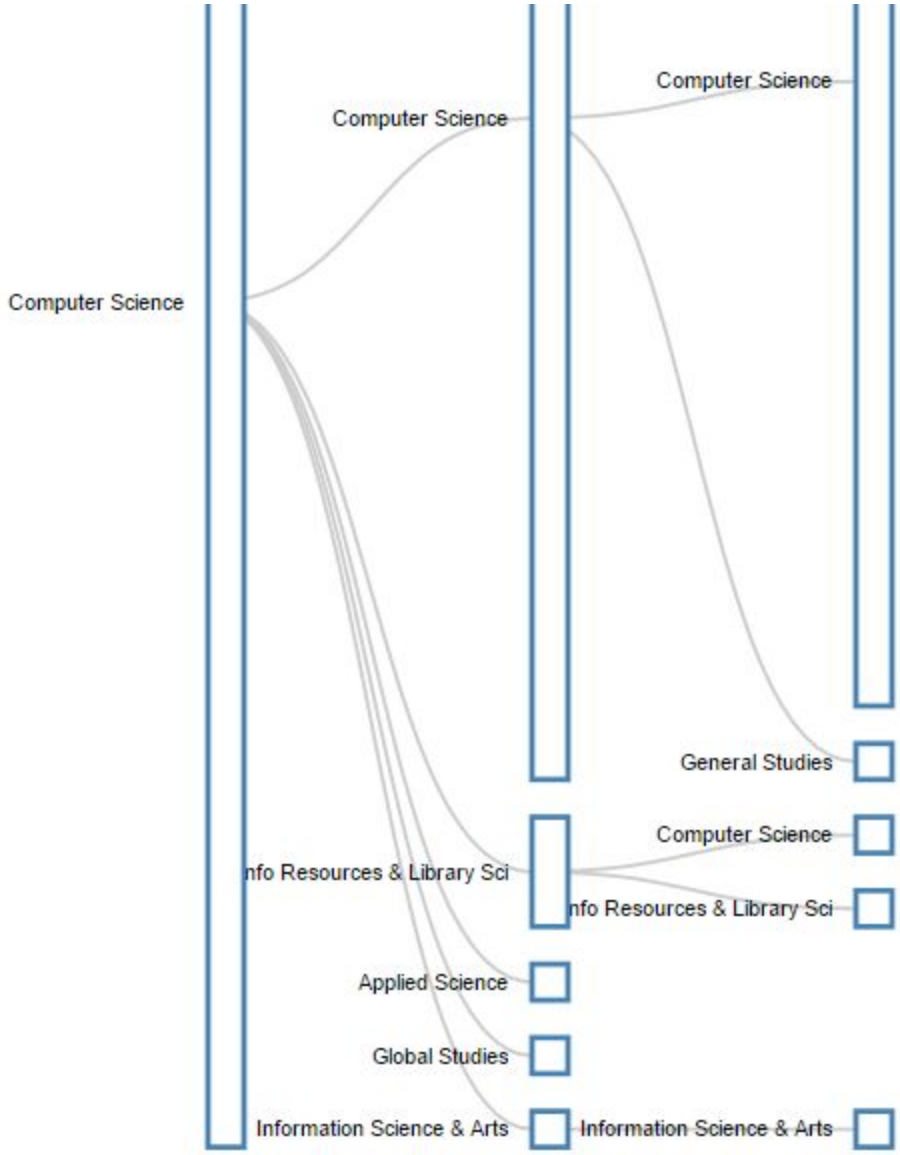

Figure 37. Text and link positions shifted to the center of each node.

Adding a filter based on count of students in each node

We first implemented this by looking at all the nodes and seeing if the number of students met a certain threshold value. If a node didn't make it, then we would change the visibility of the node to hidden.

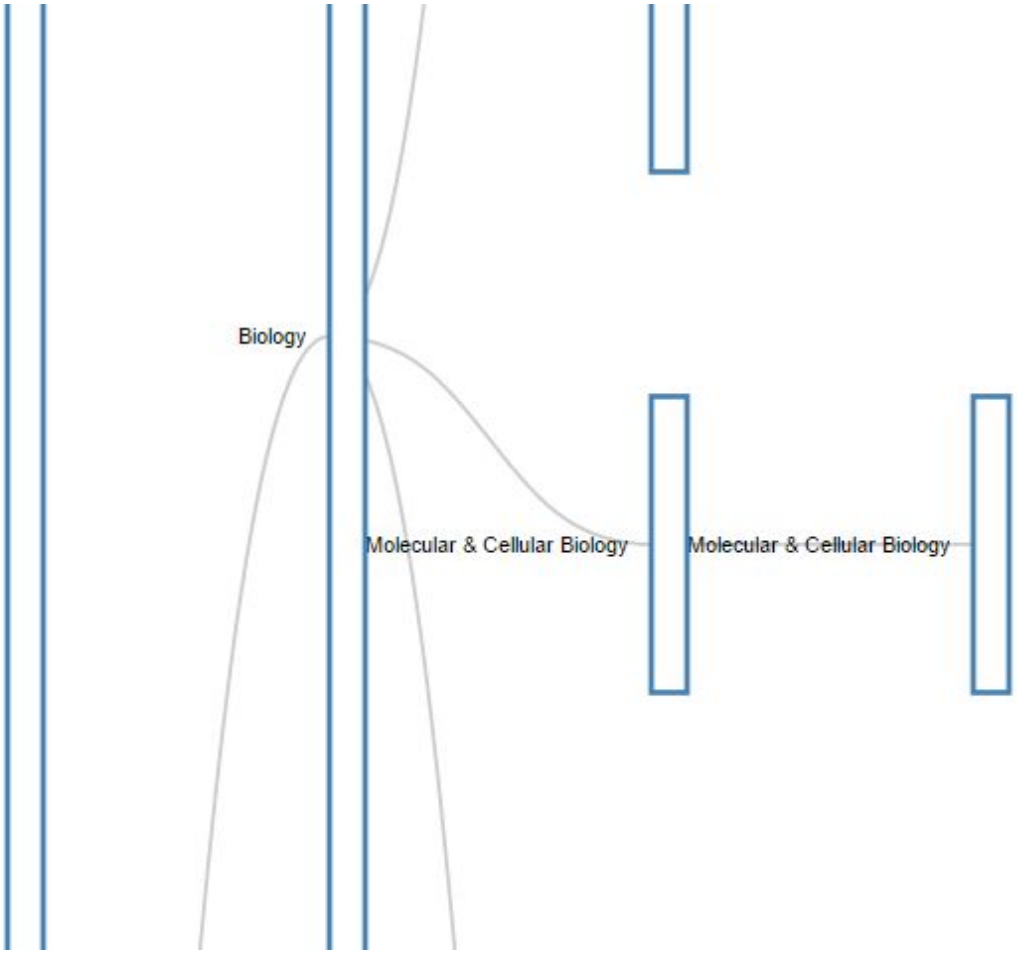

Figure 38. Biology Fall 2010 with nodes greater than 2. There are very wide spaces between nodes because of the hidden nodes.

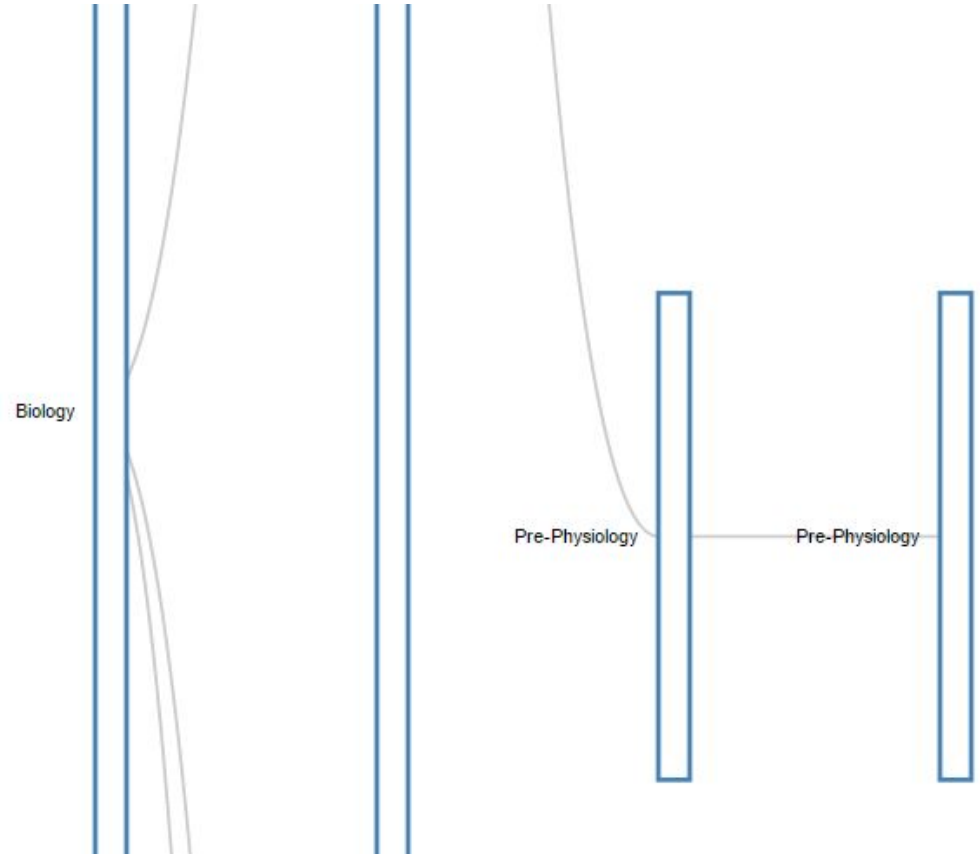

Figure 39. Biology Fall 2010 with nodes greater than 2. There are very wide spaces between nodes because of the hidden nodes.

Then we realized this probably isn't very useful, so instead of just hiding a node we take it out of the tree. After building the whole tree data structure, we had to recursively look at every node and remove them from the tree as needed.

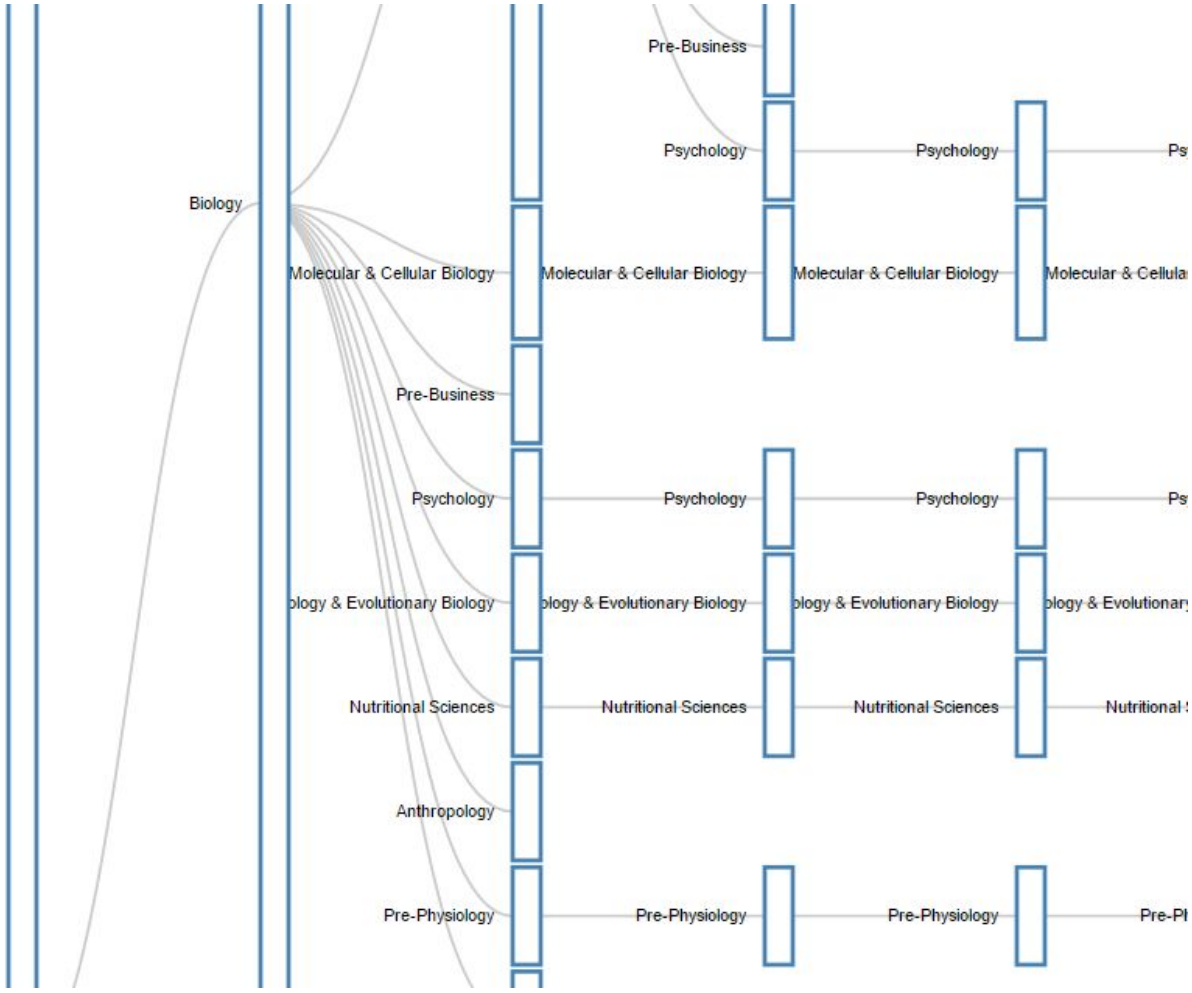

Figure 40. Biology Fall 2010 with nodes greater than 2. The re-structured data structure representation is much cleaner and probably more useful since there isn't as much space between nodes.

The purpose of rescaling the node size given a threshold greater than 1 is to conserve space. If we only want to see nodes that have a value greater than 10, then the node size does not need to be 11 ghost nodes big. Instead, it can be the smallest it needs to be, which is normally the size of a node if it is 1 ghost node big. This modified by taking the count of a leaf node and dividing it by our scale. We are left with the number of ghost nodes that should be attached to the leaf. Modifying the leaves in turn modifies the rest of the tree.

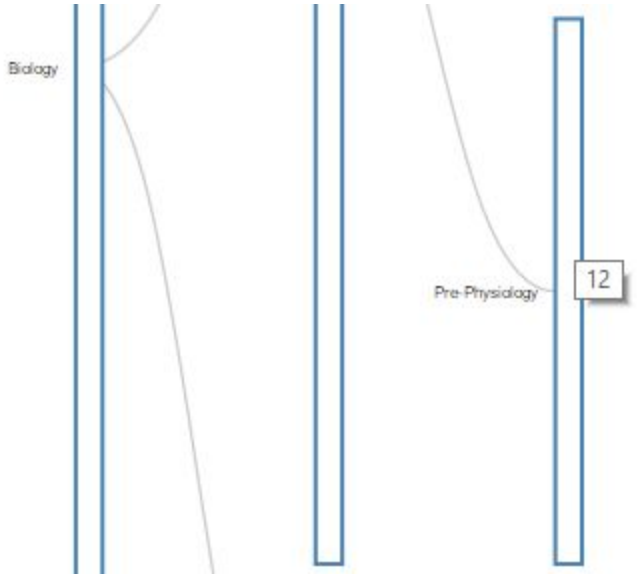

Figure 41. Biology Fall ‘10 with nodes greater than 10 without scale. Here the smallest node size is 12 for Pre-Physiology. The node looks big.

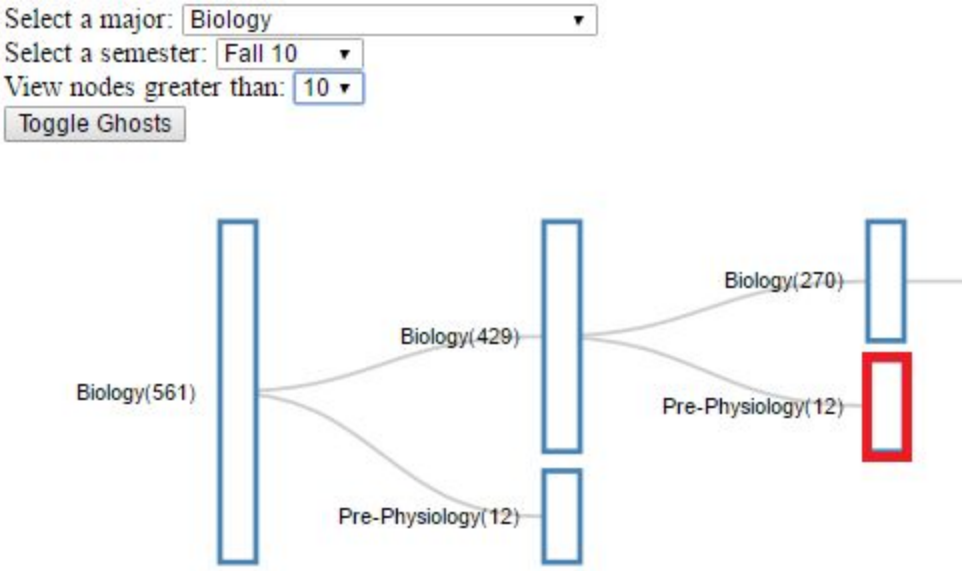

Figure 42. Biology Fall ‘10 with nodes greater than 10 with scale. The box is red is the same node as viewed in Figure 41. Here the node sizes are more appropriate since we only want to view nodes with values greater than 10.

Interactivity

Aside from the major, semester, and node filters that are mandatory, other filters are implemented to utilize the dataset to its potential. These filters allow the user to select the subset of students they are interested in examining further. The optional filters include other options like gender, ethnicity, residency status, academic program, and age range. To get the subset specifications, we took the subset and ran it through each filter using conditionals. To get the age range bar, we implemented noUISlider that we found on the internet [10].

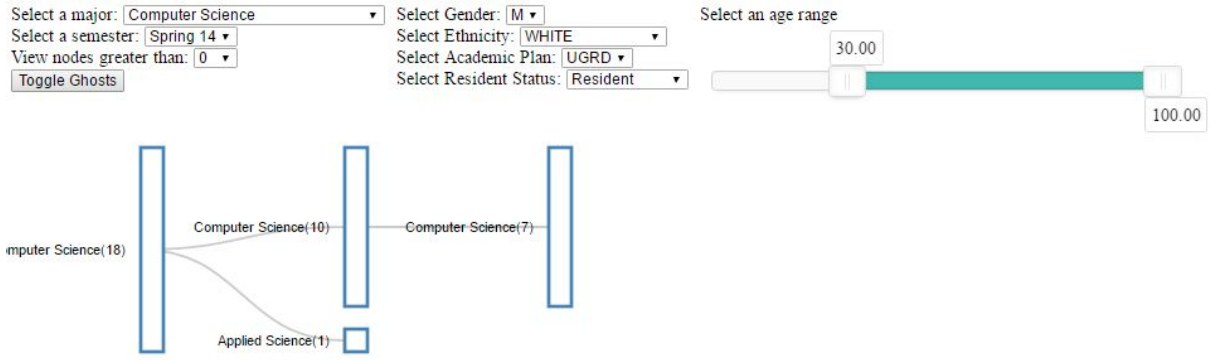

Figure 43. Demonstration of filters.

Improving the load time

We used node.js to create separate JSON files for each nested data structure in hopes of increasing the speed of the load time [9]. Instead of loading the data as an entire CSV file then creating 3 separating nested data structures, we can save the nested data structures as JSON files and load from there. This could not be done using normal Javascript because Javascript cannot write files.

Working with Node.js

We used the npm to install d3 and wrote a script to create 3 separate files for the 3 nested data structures that we use within the tree diagram.

```
$ node test/test.js
Hello world!
File exists
Parsed csv file
Success!
Success!
student_data.txt was saved

semester_data.txt was saved
```

Figure 44. 2 out of 3 files were created after the initial file was.

We only had to run the script 1 or 2 times to write all the files that we wanted.

Loading JSON files instead of nesting

In tree.html, we changed the loading of the data so that instead of loading in the CSV file of the student data and creating 3 different nested data structures, we load 3 different nested data structures from the 3 JSON files that we already created. A problem that we ran into was dealing with the asynchronous timing of loading the 3 files when the rest of the code depends on the loaded data.

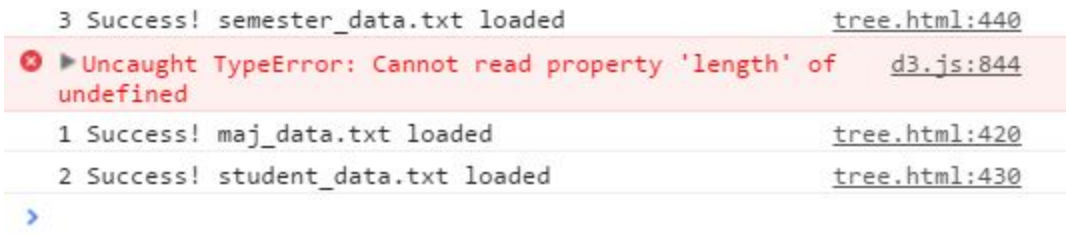

Figure 45. As evidenced by the error message, the part of the program that depends on the data was executing before the data was loaded.

To solve this, we tried implementing a d3 queue where the data would load asynchronously upon which once this was finished, the rest of the code would execute. We recorded the time to load and it seems that maybe the old time is better than the new one. The longer execution time is the result of such large data files. Because this was the cause, data normalization could possibly cut down on the loading of redundant attributes. We decided to apply data normalization to the original CSV file instead of having the program load 3 different JSON files.

For the data normalization, we created a new csv file that replaces the data that is repeated many times throughout the file with integers. That data is mapped to an integer in a JSON file. We did this for academic plan, semester, major, and ethnicity since those contained longer strings.

STRM	PERSON_S	plan_desc	acad_care	Ethnicity_	GENDER_(	UA_AGE	UA_IPEDS_RESIDENCY
0	122310	0	0	0	M	29	N
1	667811	1	1	1	F		R
2	635648	2	1	2	M	22	R
0	648491	3	1	1	F	18	R
3	914245	4	1	2	M	19	N
4	642671	5	1	1	M	19	R
5	1128358	6	1	3	F	22	R
4	653227	7	1	2	M	21	N
2	661382	8	1	2	F	20	R
2	98343	9	0	4	M	32	N
6	562393	10	0	1	F	55	N
7	873318	11	0	0	M	36	N
7	1133122	12	1	2	F	18	N
6	457259	10	0	2	F	55	R
1	753789	13	1	1	F		N
8	682628	13	1	2	M	21	R
3	900102	14	1	1	F	19	N
2	672726	12	1	2	M	19	R
3	657468	15	1	2	F	21	R
2	760854	8	1	5	M	20	N
2	806656	12	1	3	M	18	R
7	1199469	16	0	1	M	29	N
6	761599	17	1	1	F	20	R
8	682669	18	1	2	M	28	R

Figure 46. Normalized CSV file.

Figure 47. The JSON file for ethnicity.

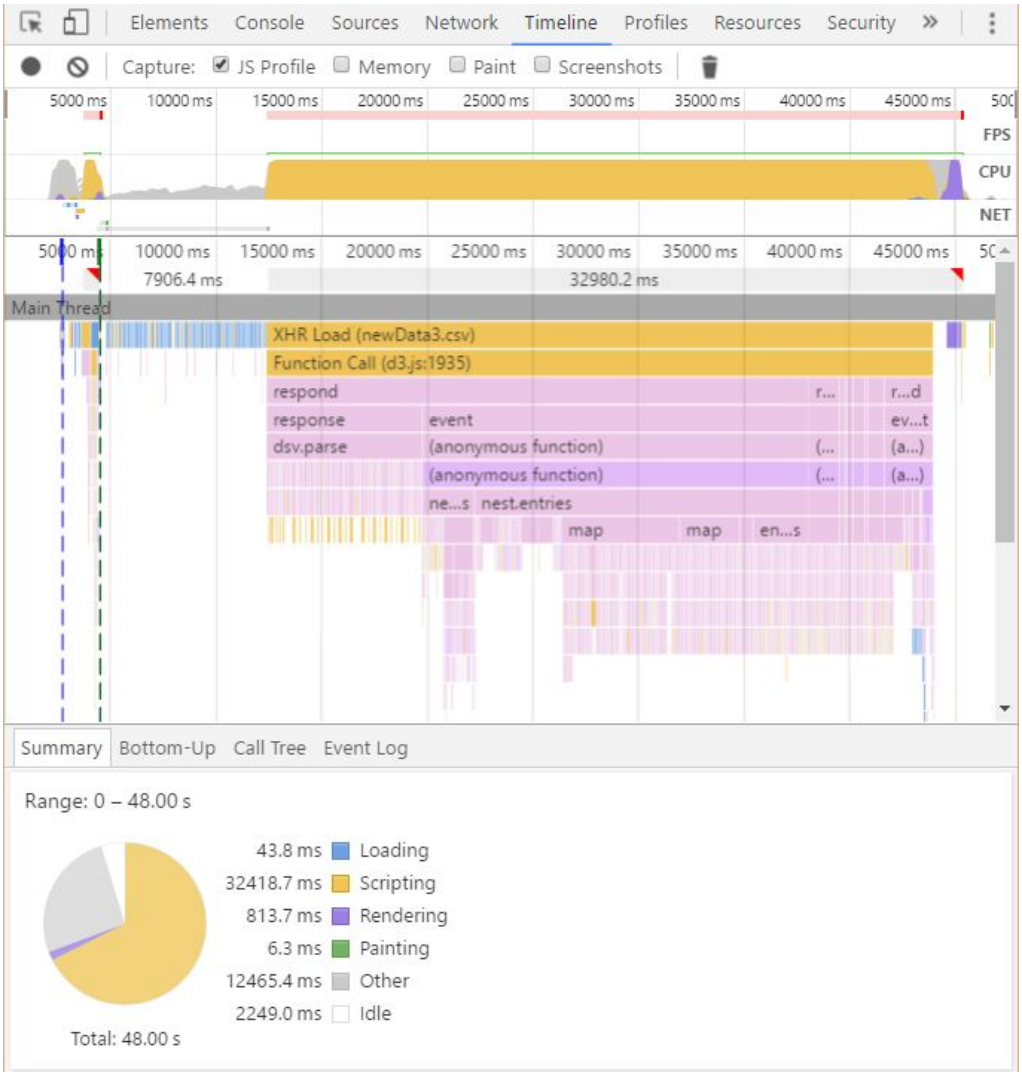

Figure 48. Tree 1 (original loading of data). This one takes a long time maybe because it's the first time we load the page.

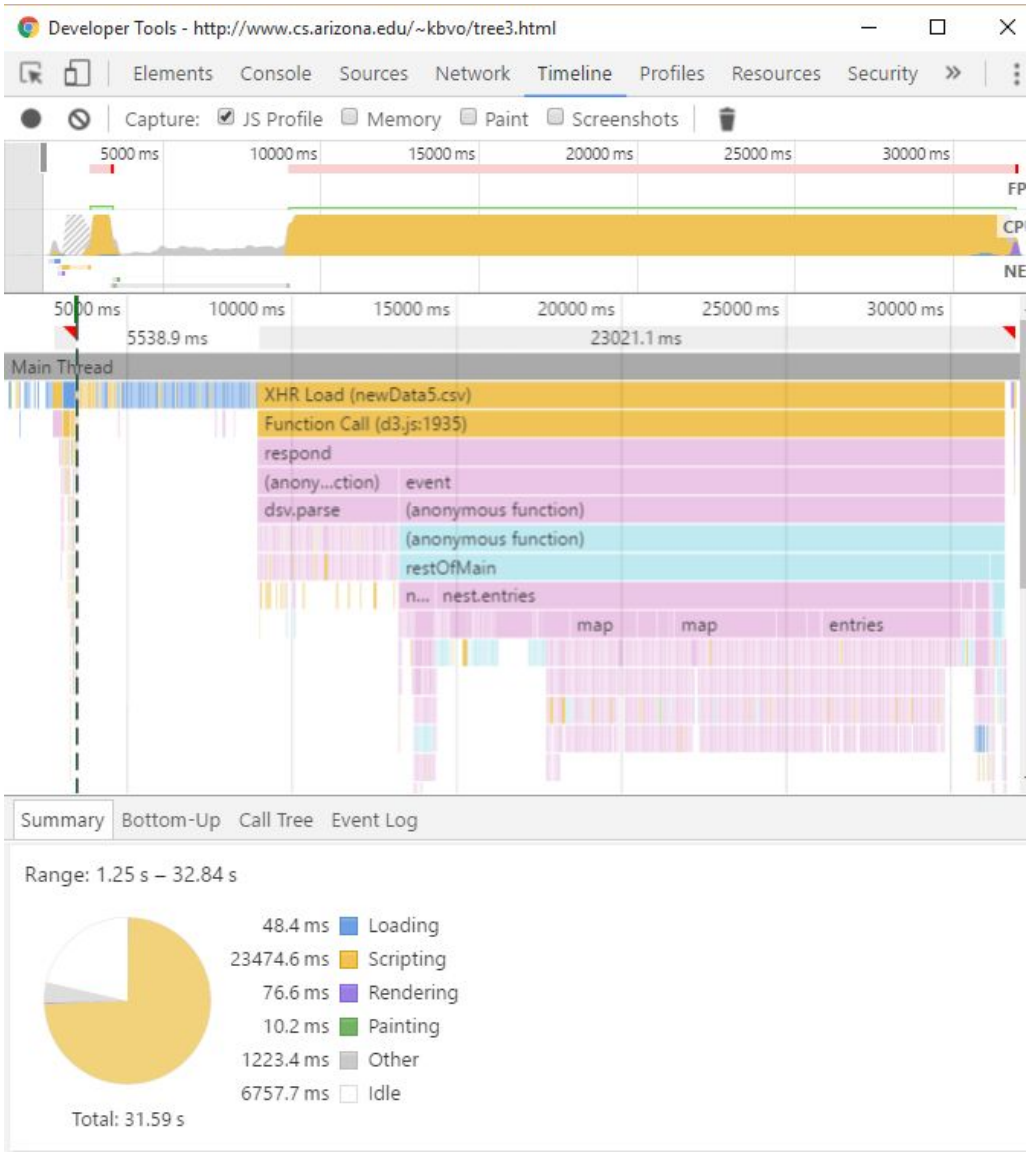

Figure 49. New loading of data. Also takes longer than normal probably because it's the first time we load the page.

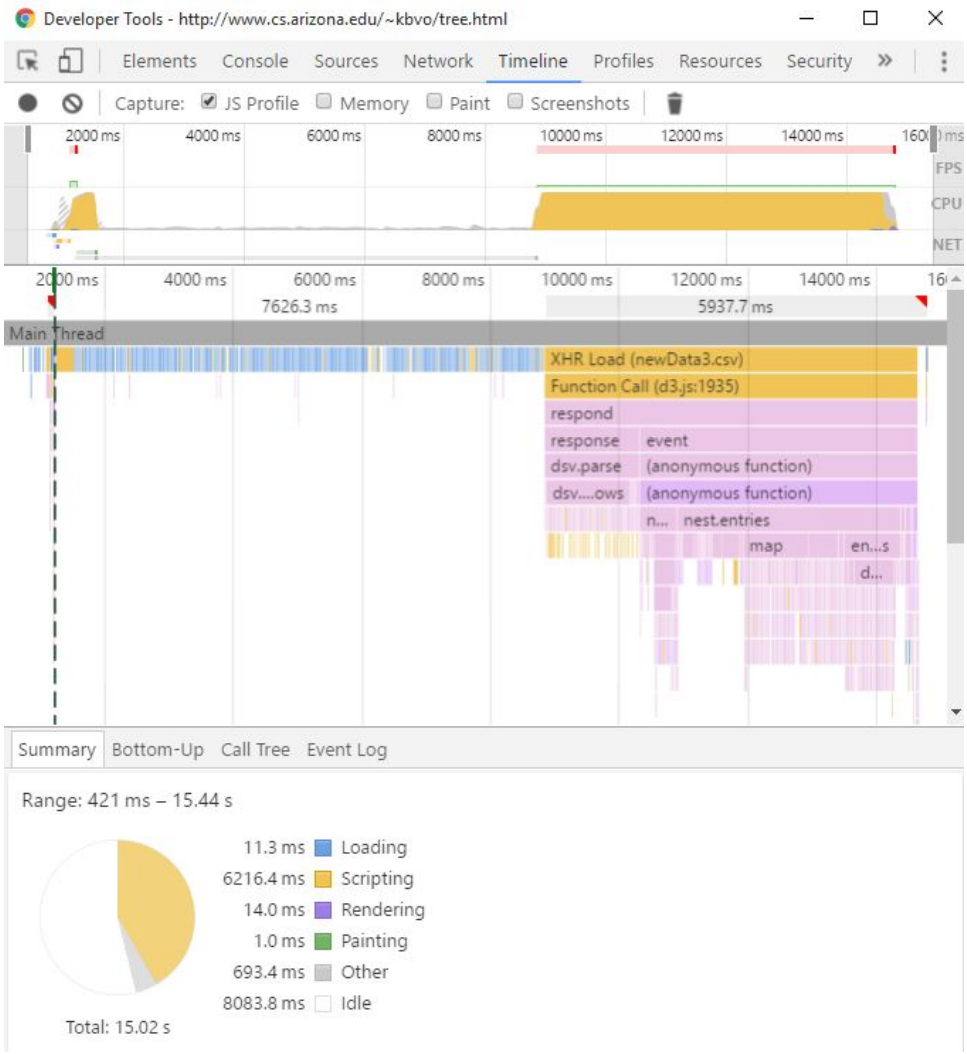

Figure 50. Original loading of data loading it for maybe the 3rd time. The loading is considerably faster and this is with disabled caching.

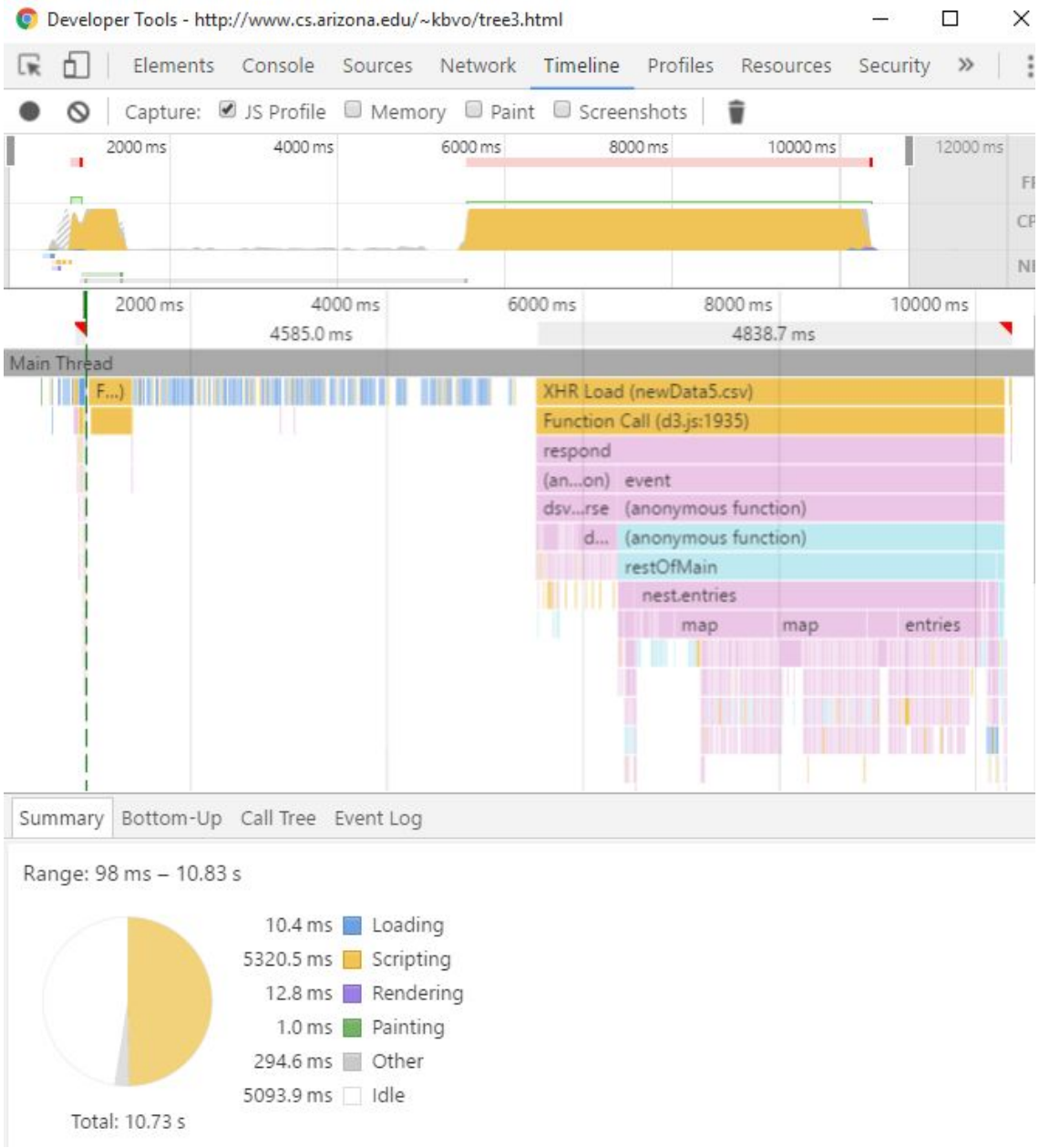

Figure 51. New loading of data, also probably the 3rd time loading the page. This is also considerably faster than the first time. The new loading of data reduces the load time of the page by a couple seconds--about 2-3 seconds faster than Figure 50.

Overall, the results varied across a large range, but I think the new loading of data improved the runtime by a couple of seconds.

Results/Findings

The resulting line graphs include the number of students in each major for each semester. Additional graphs include splittings of student demographic data that include gender, ethnicity, and residency status. Users can select which major they are curious to examine using the dropdown menu which is populated with majors from the University of Arizona.

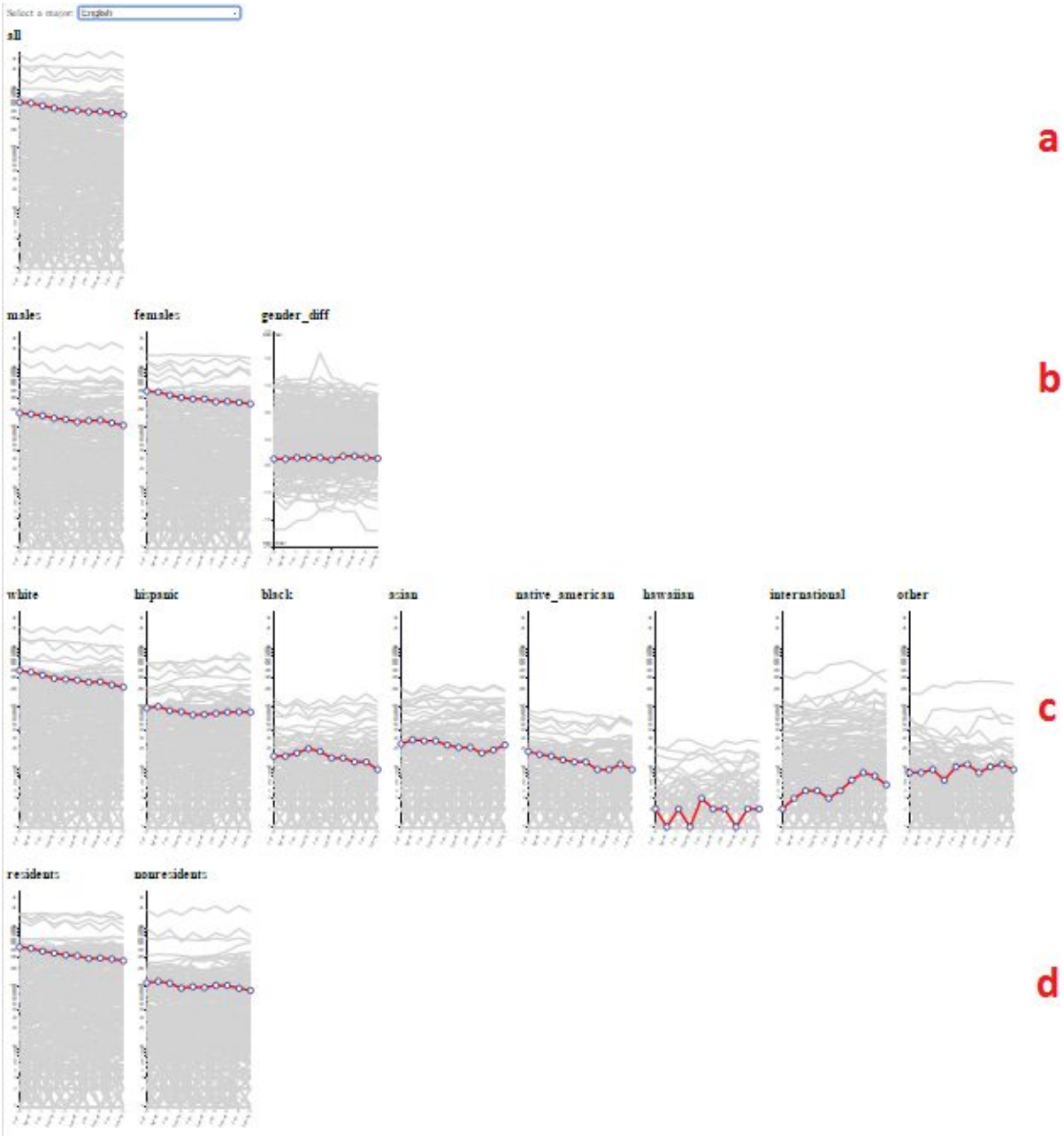

Figure 52. Graphs are grouped into divs that describe the characteristic the graph is splitting by. (a) This graph includes all students. (b) The gender group includes the male and female graphs,

along with the additional gender ratio graph. (c) These ethnicity groups include ethnicities that the UA uses for its demographic data. (d) Each student is also classified into resident and non-resident groups. Because the English is selected in the dropdown menu, this line is highlighted in red across all the graphs.

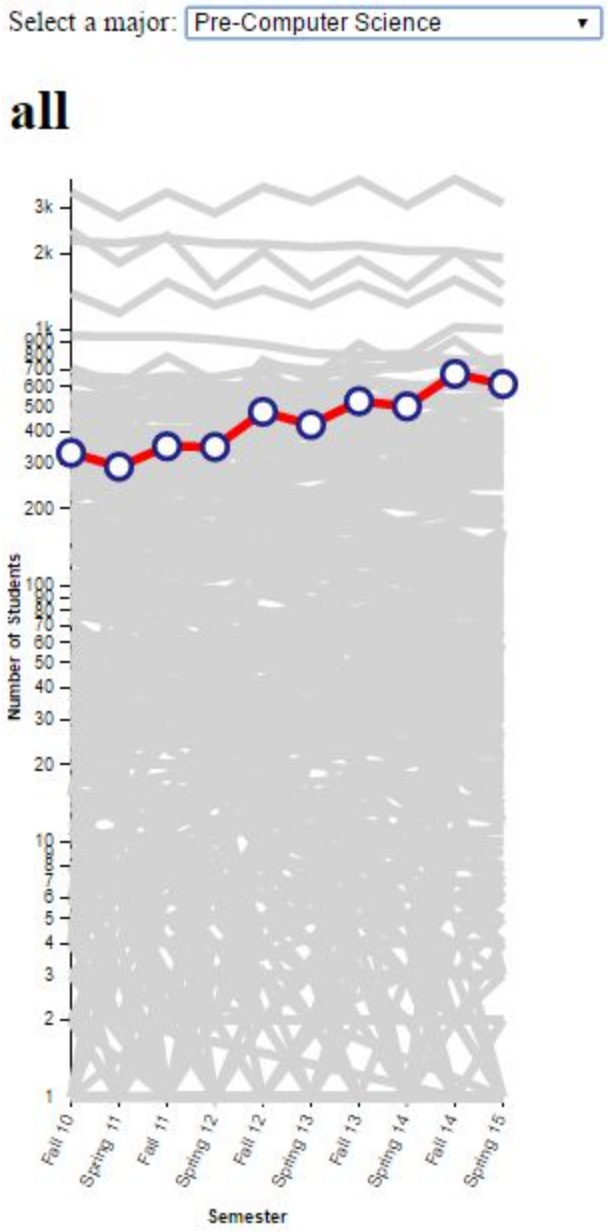

Figure 53. This is the graph for all students in the dataset at the UA. The line highlighted in red tells us that this is the Pre-Computer Science major as indicated by the selection in the dropdown menu. We can see that the number of students within this major is steadily increasing. From Fall 2010 to Spring 2015, the number of students nearly doubles. Because this is a log y-axis, increases further up the y-axis are much more significant than increases lower on the graph.

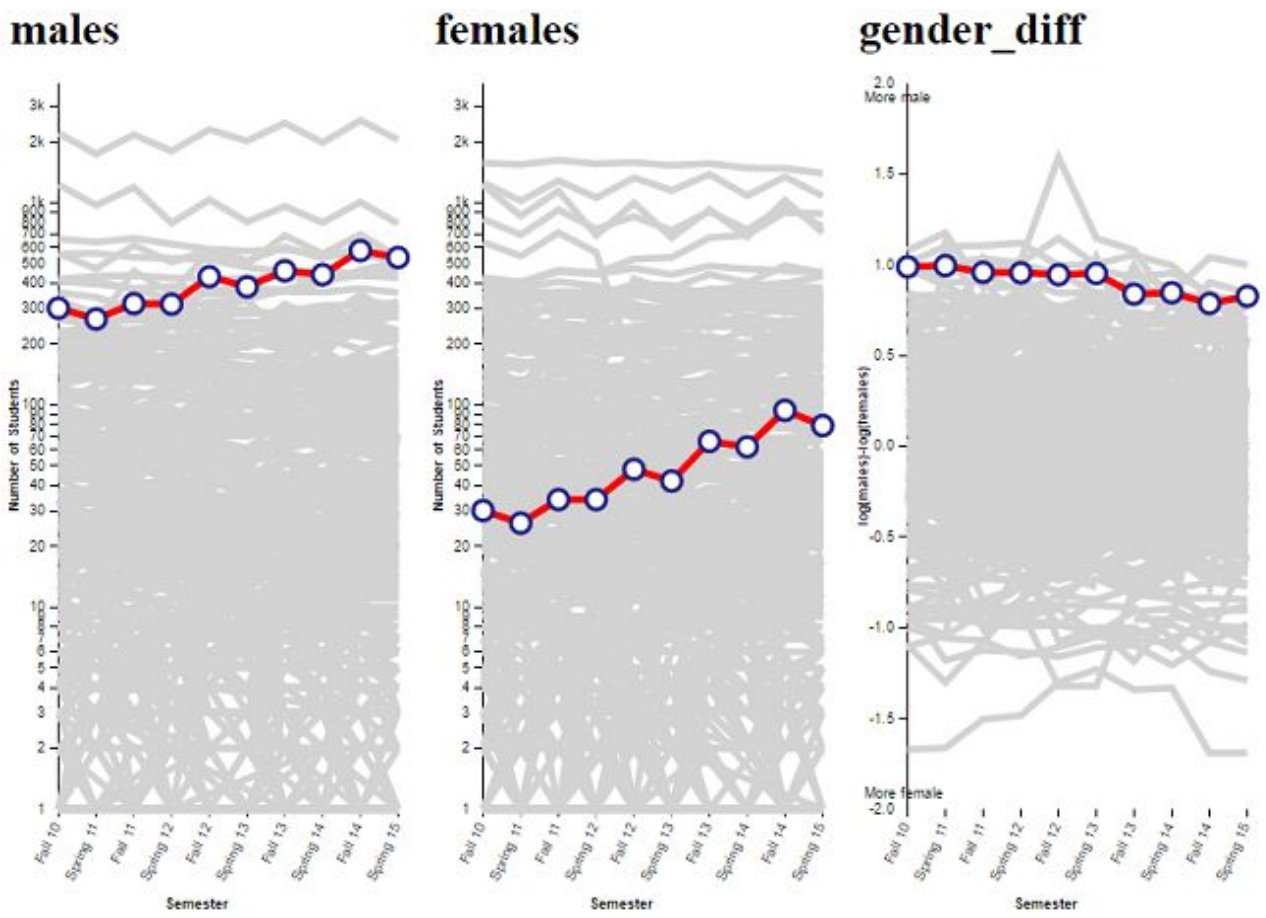

Figure 54. These are the gender graphs for Pre-Computer Science. During the Fall 2010 semester, we can see there are nearly ten times the amount of males than females. However with each increasing semester, the gap is closing. This is also illustrated in the gender_diff graph. The ratio of males to females is decreasing.

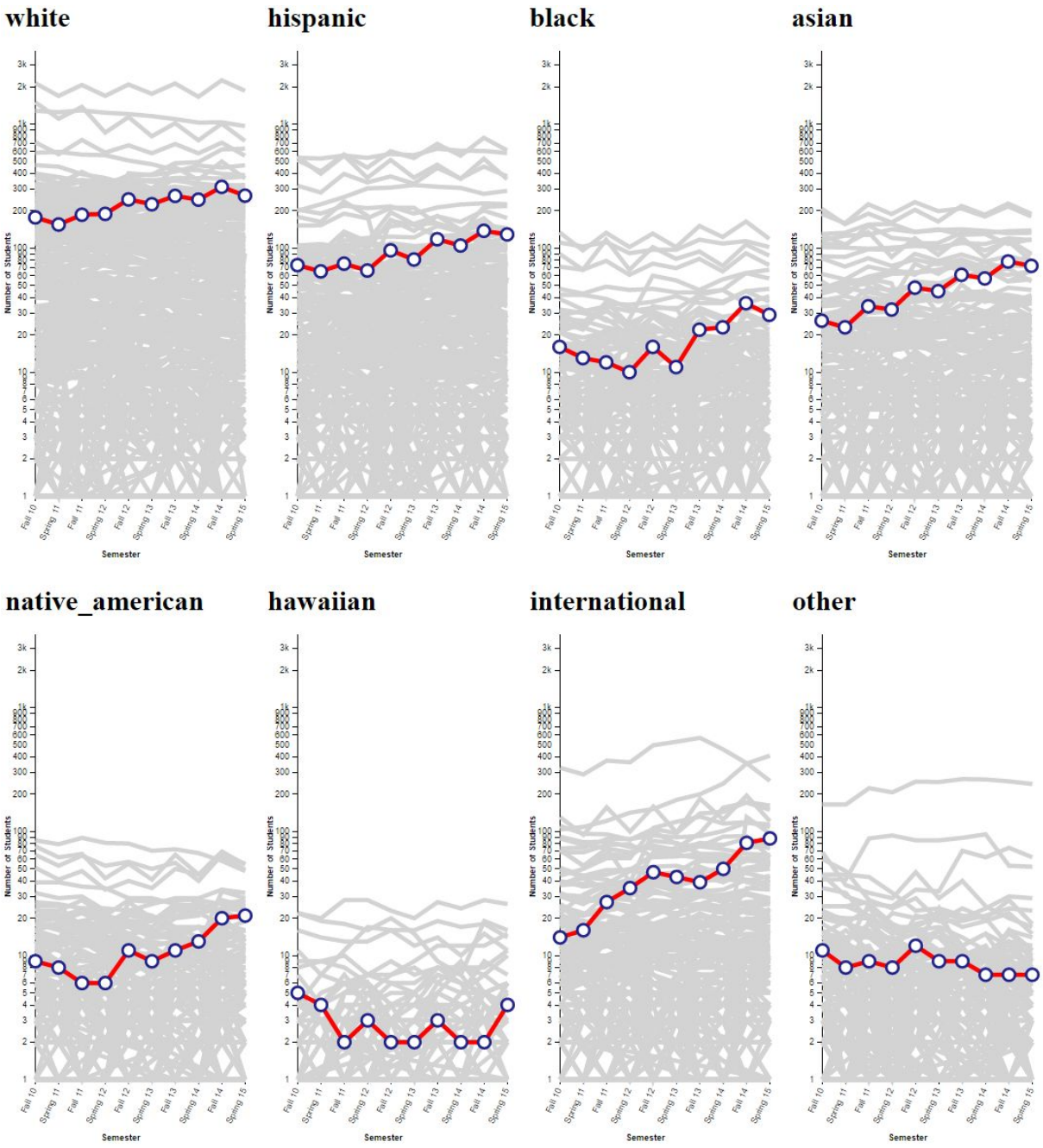

Figure 55. As reflected in nearly all graphs for ethnicity, the Pre-Computer Science major is growing.

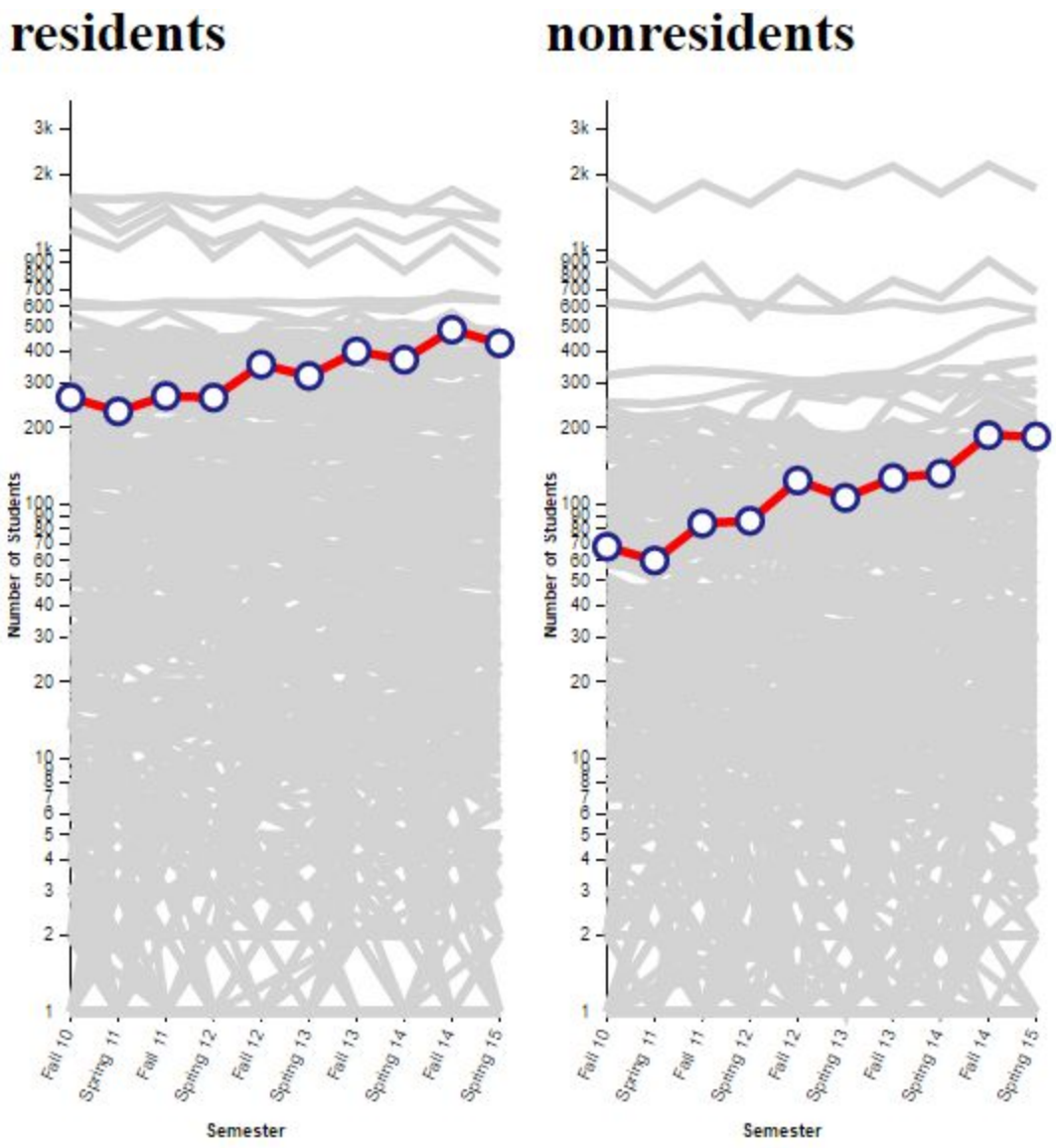

Figure 56. The number of non-residents majoring in Pre-Computer Science about doubled from Fall 2010 to Spring 2015, whereas Residents increased around 150%.

Through a quick lookup of the Pre-Computer Science major, these line graphs clearly communicate that the major is rapidly growing, there are approximately 100 males per every female within the major, and the growth is present in nearly all ethnicities regardless of residency status.

The resulting tree diagram allows user to examine the major path of students based on the subset of students the user selects. This subset of students makes up the root of the tree diagram on the far left while each level of the tree represents the subsequent semester. The starting subset can be manipulated through a number of filters specified at the top. Each set of nodes in a level is

a subset of its parent's set of students. There are 4 filters that must have a value set at all times. These are the major, semester, nodes that contain at least a number of students, and the age range. The optional filters that may be empty, (and empty meaning all the listed options are included), include gender, ethnicity, academic plan, and resident status.

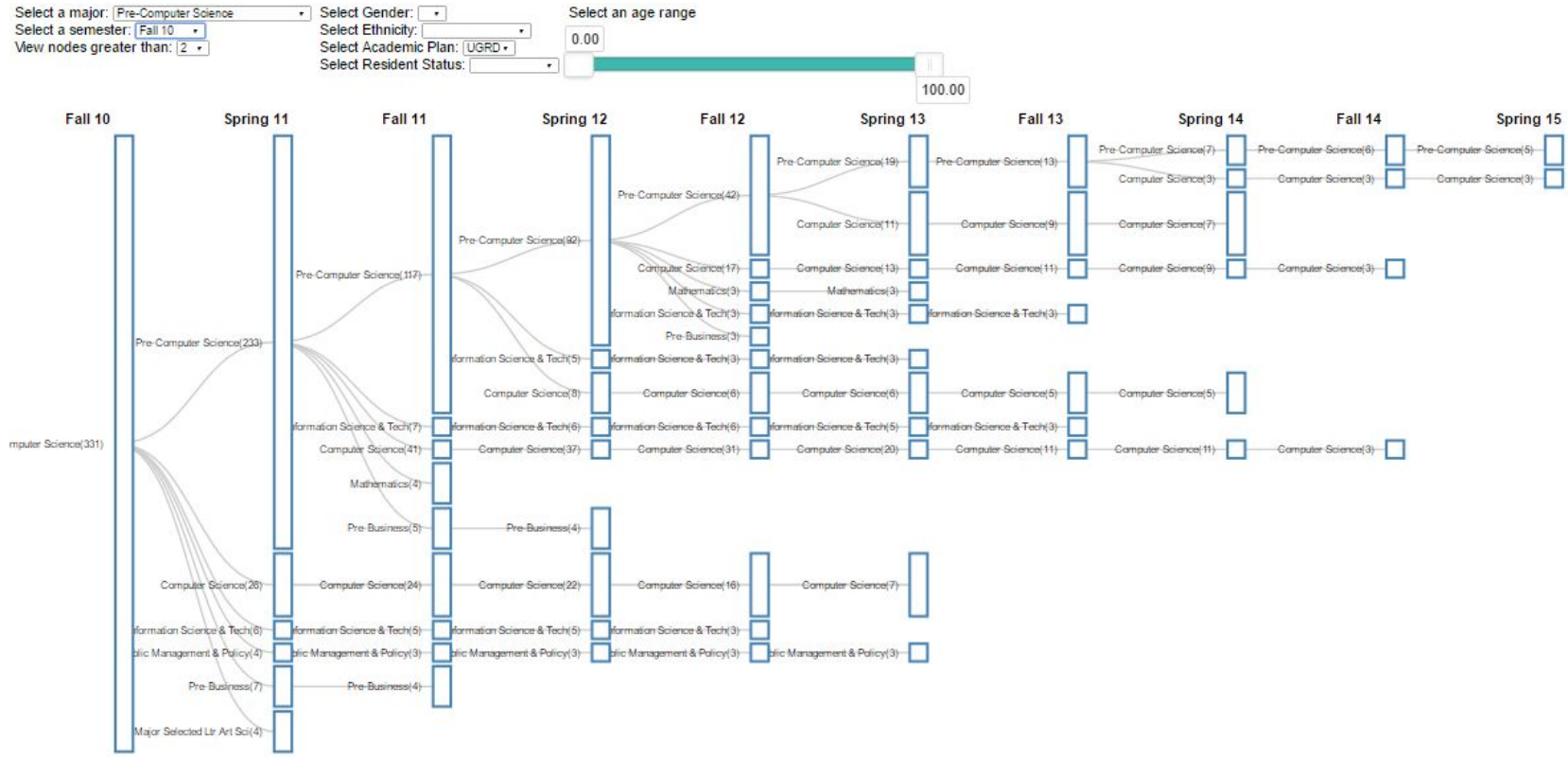
Select a major: Pre-Computer Science
Select a semester: Fall 10
View nodes greater than: 2
Select Gender:
Select Ethnicity:
Select Academic Plan: UGRD
Select Resident Status:
Select an age range
0.00
100.00
Fall 10
Spring 11
Fall 11
Spring 12
Fall 12
Spring 13
Fall 13
Spring 14
Fall 14
Spring 15
mputer Science(331)
Pre-Computer Science(233)
Computer Science(28)
formation Science & Tech(6)
blic Management & Policy(4)
Pre-Business(7)
Major Selected Ltr Art Sci(4)
Pre-Computer Science(117)
formation Science & Tech(7)
Computer Science(41)
Mathematics(4)
Pre-Business(5)
Computer Science(24)
formation Science & Tech(5)
blic Management & Policy(3)
Pre-Business(4)
Pre-Computer Science(92)
formation Science & Tech(5)
Computer Science(8)
formation Science & Tech(6)
Computer Science(37)
Pre-Business(4)
Computer Science(22)
formation Science & Tech(5)
blic Management & Policy(3)
Pre-Computer Science(42)
Computer Science(17)
Mathematics(3)
formation Science & Tech(3)
Pre-Business(3)
formation Science & Tech(3)
Computer Science(6)
formation Science & Tech(6)
Computer Science(31)
Computer Science(16)
formation Science & Tech(3)
blic Management & Policy(3)
Pre-Computer Science(19)
Computer Science(11)
Computer Science(13)
Mathematics(3)
formation Science & Tech(3)
formation Science & Tech(3)
Computer Science(6)
formation Science & Tech(5)
Computer Science(20)
Computer Science(7)
Pre-Computer Science(13)
Computer Science(9)
Computer Science(11)
formation Science & Tech(3)
Computer Science(5)
formation Science & Tech(3)
Computer Science(11)
Pre-Computer Science(7)
Computer Science(3)
Computer Science(7)
Computer Science(9)
Computer Science(5)
Computer Science(11)
Pre-Computer Science(6)
Computer Science(3)
Computer Science(3)
Computer Science(3)
Pre-Computer Science(5)
Computer Science(3)

Figure 57. This is the subset of students that were listed as Pre-Computer Science in Fall 2010. Nodes that have less than 3 students are excluded. Looking each level, the most common majors students switch to from Pre-Computer Science are Computer Science, Information Science & Tech, Pre-Business, and Mathematics.

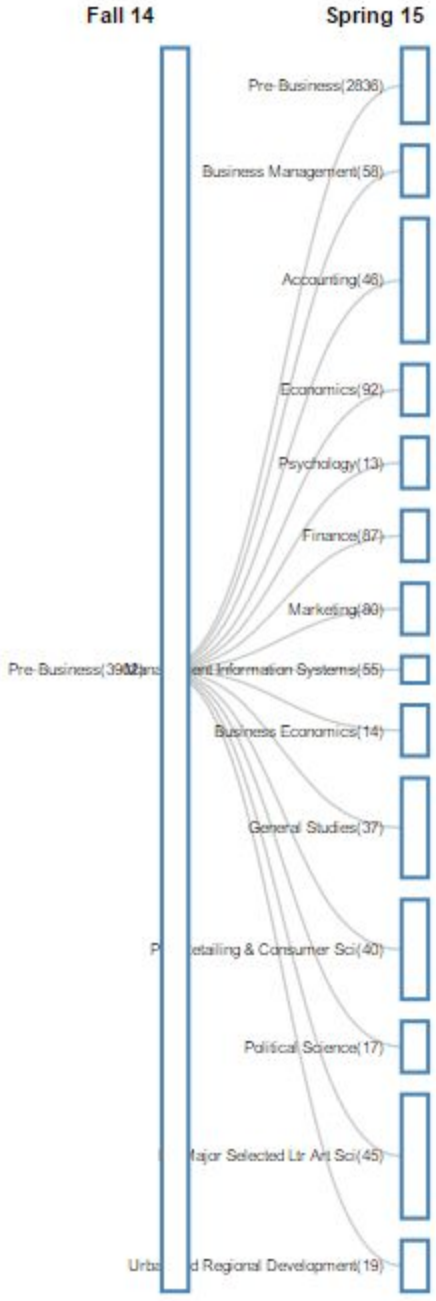

Figure 58. This is the subset of students that were listed as Pre-Business in Fall 2014. Nodes that have less than 11 students are excluded. Because Pre-Business has the most number of students (as found from exploring the line graphs), it makes sense that there are a greater

number of students switching to a different major. Most majors these students switch into are more focused majors within Eller whereas others move into different disciplines entirely (Psychology, Political Science, etc.).

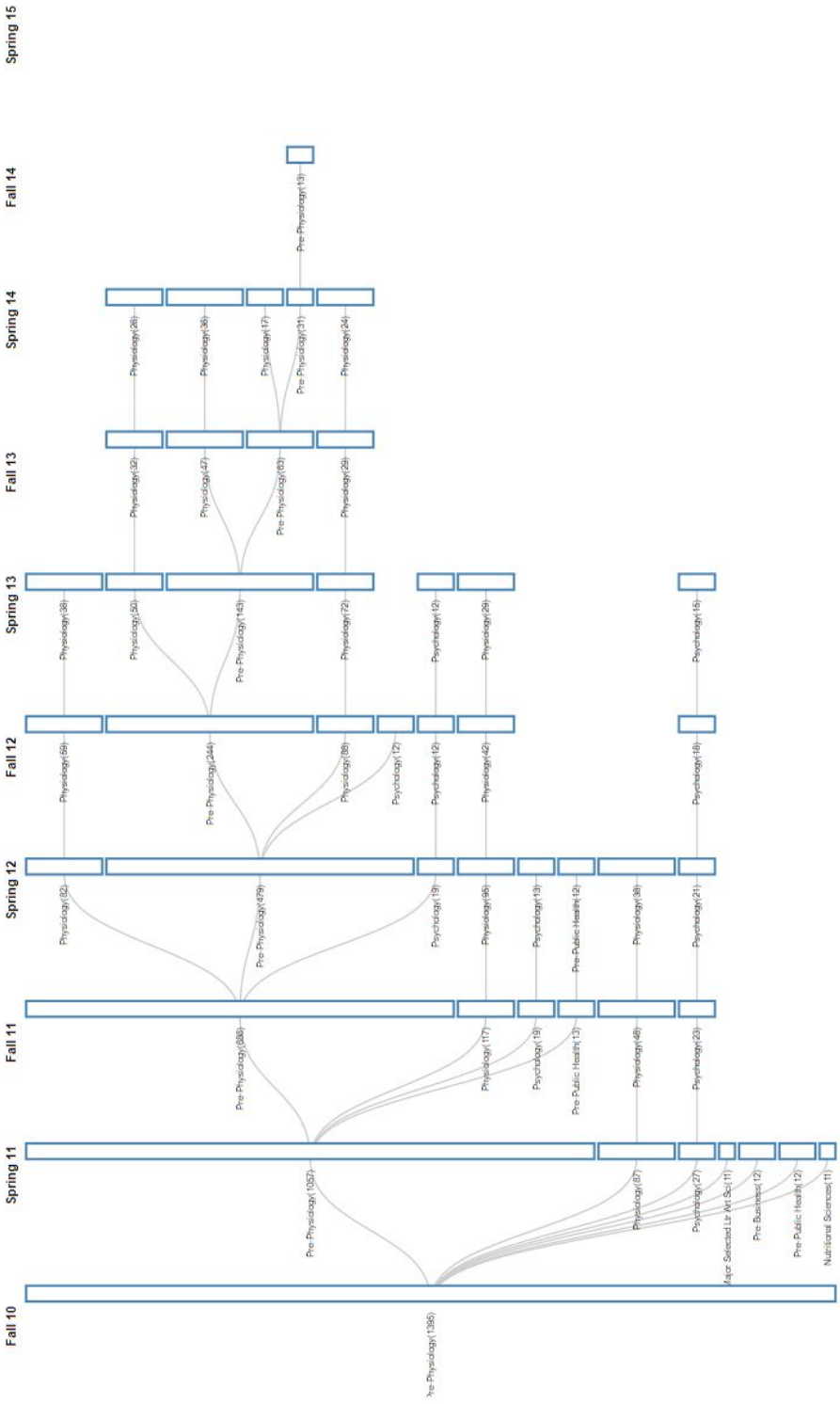

Figure 59. This is the subset of students that were listed as Pre-Physiology in Fall 2010. Nodes that have less than 11 students are excluded. As with the pre-majors, there appears to be a lot of movement. For Pre-Physiology students, Physiology, Psychology, and Pre-Public Health are popular majors to switch into.

Using the tree diagram visualization as a tool to figure out where people from specific majors is very effective. Although the size of the nodes are not exactly proportional to the number of students that the node represents, the relative size of nodes compared to the size of the root node gives users a rough idea of how many people decide to stay and leave.

Discussion

Working with d3 taught us how much this library has to offer in terms of what you can build with it. There was a very steep learning curve throughout building these visualizations. At first, the challenge was how to obtain the aggregates of students given a specific set of characteristics. A lot of times, d3's built-in methods would return an object with multiple levels of nesting to which we did not know how to get to a specific level of nestedness within that data structure. However in those situations, we could use our knowledge of Javascript to help. Another challenging aspect of working with d3 was how to draw the visualizations that went beyond the default, built-in layout. This required us to look at how the built-in layouts were constructed and figure out what needed to be changed for the visualizations to look the way we wanted it to.

There were a number of other features we did not have time to implement. To make the visualizations more effective, it would have been better to allow users to compare 2 different diagrams side-by-side. Another helpful feature would be using colors to color each node based on major so that users can easily identify popular majors. Colors can also be used as a tool to spot the number of students that a node represents. A node with darker color can represent a node with a greater number of students whereas a node with less students can be a lighter shade of the same color. If we had the data regarding which college each major was a part of, colors could be used to denote the different colleges. Input from people who might be interested in using this tool would also be useful in finding out what additional features could have been added. Looking at a subset of students across all time for a major would have probably been the most valuable feature. This allows for a better idea of the proportion of students that move if the numbers across all semesters were aggregated.

Another point of weakness that we did not have time for was coming up with a way to make it so that the number of students within a node was reflected using the size of a node. As it stands now, the size of a node is determined by the number of students contained in a node's leaves.

Conclusion

In completing this project, I was able to answer my question of what the progression of a student that begins in a specific major typically looks like. The path does not necessarily look the same for all students across time; however I was able to get a general idea of how certain students move throughout the university. Something that struck me was how each student's path is truly unique. Throughout each diagram I examined, I was not expecting the amount of students who were the only ones to switch to a specific major for a semester. Both the tree diagrams and the line graphs gave me a greater understanding of how students are grouped at the university.

References

[1] Aisch, Gregor, Robert Gebeloff, and Kevin Quealy, “Where We Came From and Where We Went, State by State,” *New York Times,* August 19, 2014, accessed May 3, 2016, http://www.nytimes.com/interactive/2014/08/13/upshot/where-people-in-each-state-were-born.html.

[2] Cleveland, William S. *The Elements of Graphing Data*. Hobart Press, 1994.

[3] “Data-Driven Documents,” accessed May 3, 2016, https://d3js.org.

[4] “Java Platform, Standard Edition 7 API Specification,” accessed April 29, 2016, https://docs.oracle.com/javase/7/docs/api/.

[5] Minard, Charles Joseph, “Carte Figurative des pertes successives en hommes de l'Armée Française dans la campagne de Russie 1812-1813” Paris 1869.

[6] “Multi-Line Voronoi,” accessed February 12, 2016, last modified February 8, 2016, https://bl.ocks.org/mbostock/8033015.

[7] “Multi-Series Line Chart,” accessed May 3, 2016, last modified April 29, 2016, https://bl.ocks.org/mbostock/3884955.

[8] Munzner, Tamara. *Visualization Analysis and Design*. CRC Press, 2014.

[9] “node.js,” accessed April 12, 2016, https://nodejs.org/en/.

[10] “noUiSlider: JavaScript Range Slider,” accessed April 19, 2016, http://refreshless.com/nouislider/.

[11] “Ordinal Scales: Categorical Colors,” accessed May 3, 2016, last modified December 9, 2014, https://github.com/mbostock/d3/wiki/Ordinal-Scales#category10.

[12] “Quantitative Scales: Log Scales,” accessed May 3, 2016, last modified November 23, 2015, https://github.com/mbostock/d3/wiki/Quantitative-Scales#log.

[13] “Tree diagrams in d3.js,” accessed May 3, 2016, last modified January 11, 2014,

http://www.d3noob.org/2014/01/tree-diagrams-in-d3js_11.html.

[14] “Tree Layout,” accessed April 29, 2016, last modified October 9, 2014, https://github.com/mbostock/d3/wiki/Tree-Layout.