Visualizing UW Course Prerequisite Sequences

Dylan Babbs
University of Washington
dbabbs@uw.edu

Jordan Starkey
University of Washington
jds56@uw.edu

ABSTRACT

The UW Course Prerequisite Explorer provides an intuitive experience to examine prerequisite course sequences. Navigating the University of Washington course catalog can be a difficult task – especially when dealing with prerequisites. Currently, the course catalog lists the prerequisites necessary in order to enroll in a specific course, however, the description lists only the first degree of prerequisites required. The goal of this tool is to improve degree and coursework planning transparency by providing an intuitive visualization experience using trees. The only required action from the user is to input a course in order to explore prerequisite sequences. The absence of any tool of this capacity currently forces students to backwards trace their course sequences to find course's second (or higher) degree prerequisites.

INTRODUCTION

Registration at the University of Washington occurs on a quarterly basis. Course offerings are listed in the official University of Washington course catalog. The course catalog maintained by the university is rather comprehensive; each course contains information regarding the title, credit count, credit type, description, and required prerequisites. All this information is very practical to a prospective student – however, only the first degree of required prerequisites are displayed. Only displaying the first degree of required prerequisites creates a difficult problem for a prospective student. For example, CSE 373: Data Structures and Algorithms lists CSE 143 as the only required prerequisite. Unbeknownst to a student browsing the catalog, CSE 143 requires CSE 142 as a prerequisite. Therefore, the prerequisites needed to enroll in CSE 373 are both CSE 142 and CSE 143.

CSE 373 Data Structures and Algorithms (3)
Fundamental algorithms and data structures for implementation. Techniques for solving problems by programming. Linked lists, stacks, queues, directed graphs. Trees: representations, traversals. Searching (hashing, binary search trees, multiway trees). Garbage collection, memory management. Internal and external sorting. Intended for non-majors. Not open for credit to students who have completed CSE 332. Prerequisite: CSE 143.
View course details in MyPlan: CSE 373

Figure 1: CSE 373 course description in the official University of Washington course catalog.

In order to find if a student is eligible for the course, they would need to manually browse the catalog in reverse, searching for prerequisite requirements across the page, a difficult and tedious task. At the University of Washington, students are blocked from enrolling in certain courses they don't meet prerequisite requirements for. Required course sequence for degree programs can be found on a department's website, however, each department maintains their own website, leading to inconsistent pages and unique formatting across official University of Washington subdomains. Prospective students to a degree have no consistent and aggregated location to find suggested course sequence pathways.

The UW Course Prerequisite Explorer tool aims to improve course selection and degree planning transparency by providing an intuitive interface using tree visualizations to explore course pathways. The tool allows the user to input a course they are interested in exploring the sequence for. Upon the user's query, an interactive tree is generated with the queried course as the parent node. The node's children are the course's "post"-requisites, that is, they are the courses a student becomes eligible to enroll in once they complete the queried course. For example, a sample output for a query of CSE 142, the introductory programming class for undergraduates, would look like such:

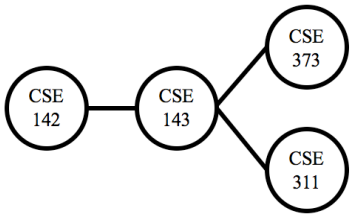

Figure 2: Sample tree output for a query of "CSE 142"

RELATED WORK

The UW Course Prerequisite Explorer was inspired by two different tools: UW Course Search (Brice Hulse) and Course Focus (iSchool Course Sector).

UW Course Search

UW Course Search [4] is a tool developed by Brice Hulse in 2016 which allows the user to compare UW course catalog offerings with Rate My Professor ratings. The project joined data scraped from the UW course catalog and data scraped from ratemyprofessor.com. The tool allows a user to query courses based off credit type, professor name, department, level, and Rate My Professor scores.

Course Focus

Course Focus [1] is a tool developed by Course Sector (a group from the UW iSchool's DataLab). Course Focus performs a very similar function to this project. Importantly, however, Course Focus' version provides no visualization of the results. Course Focus lists the prerequisites and "post"-requisites available for each course, but the results are only listed in text. By implementing a visualization of this concept, students will more intuitively be able to explore multiple course sequences, instead of the course before or after.

METHODS

Building the tool consisted of three different main areas: data collection and aggregation, client side development, and styling and layout.

Data Collection & Web Scraper

The majority of the project's development time was devoted towards data collection, aggregation, cleaning, and munging. Using a python web scraper, implemented with the Beautiful Soup library, the team scraped each department's course offerings on the official UW webpage. The department's course catalog lists all the courses taught within the program, including outdated courses no longer offered. One problem with such a comprehensive listing is that some courses appear to be no longer offered, thus making the data irrelevant. An example of this case can be seen with CSE 326. After identifying tags that contain course title and description, the team broke down the description at an even lower level to add attributes to the JSON element. These attributes included *course title*, *course description*, *credit count*, and *credit type*. Many attributes were dependent because nearly all non introductory courses require prerequisites. However, not all all course entries contained identical attribute fields. For example, the *professor* attribute was only applicable to courses in which the description contained a specific professor. Other examples of dependent attributes include: *chosen prerequisites* (those identified as either/or), *required prerequisites*, and *offered jointly* (courses that are offered jointly between different departments). Regex was used to handle the string splicing of all these elements within the data. Although this method isn't entirely foolproof, the team was happy with its accuracy. Accounting for all edge cases was a difficult task due to the sheer amount of assumptions that must be made when scraping a large amount of unknown data. A simple list of key value pairs displayed all courses within that program and their respective information.

In order to nest all these courses within each other, a recursive approach was used to search through each of the chosen courses and required prerequisites. An attribute was then added to each of the courses prerequisites that delineated the attribute as *chosen* or *required*. The attribute was then encoded by different colored links in the dendrogram. The development team decided to recurse on required prerequisites as the majority of chosen prerequisites belonged to different departmental programs; this would have resulted in an overcrowded tree. The recursive algorithm added the respective element to a course's required prerequisites attribute. Next, the algorithm searched the course elements' *required prerequisite* and added the respective elements into a combined children attribute. The combined children attribute is the attribute the D3.js code searched through in order to create nested nodes. Finally, each program was written to an external JSON file containing nested data. The JSON files are read back and merged with a program attribute to delineate between each overall list element. This method drastically improved the JSON query speed on the client side. The method searched for the program name in the search field and then returned the elements of that program to further expand on the search for the required class. The scraper's most difficult assignment to implement was the recursive approach; this task required countless trial and error as well as manual verification on the program's course catalog.

Client Side Development

The team implemented the visualization using the JavaScript library Data Driven Documents (D3.js) [2]. An example and guide created by Mike Bostock (co-creator of D3.js) of collapsible tree diagrams served as the framework of the tool's interface [3].

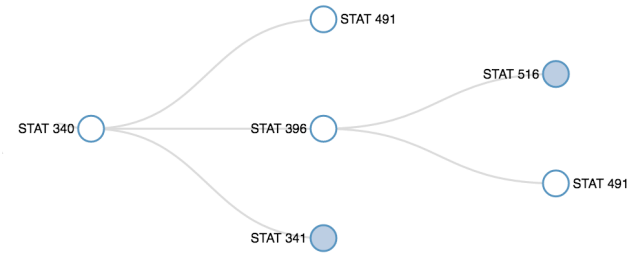

Figure 3: An example of the D3 tree layout after a STAT 340 query.

In order to maintain node hierarchy, the team elected to implement the D3 tree layout (d3.layout.tree()). This implementation is important to maintain since it gave an idea of foresight with degree planning. The tree layout is an object containing a set of node objects, each of which is characterized by the following attributes:

- *parent*: the parent node; the root node of the parent has a null value
- *children*: an array containing the children nodes; the leaves nodes have null value
- *depth*: the level of the node in the tree structure
- *x*: the x coordinate of the node
- *y*: the y coordinate of the node

The collapsible feature is important for the tool in order to initially show nodes to the first degree and allow for further exploration. Some programs had innumerable courses nested within each other that if all paths were initially shown at once, the visualization would be extremely overwhelming for the user. Certain programs, such as biology, required a scrollable canvas because the high amount of nodes cause the tree to extend off the page. The team decided to ignore the case of two nodes pointing to the same child due to the fact that the child could be on different hierarchical levels. Linking the two nodes across different levels would shift the visualization into a force graph or network layout, instead of a clean, well formatted tree diagram. The D3 data join expects that each document object model (DOM) node will correspond to a different element in the data array. In order to deal with this problem, the development team chose to have two elements in the data array assigned to the same underlying object. In the future, a potential improvement to this method would be to define a custom join key function, which by definition relies on a way to differentiate the data elements with an id or key property to extend a node to its furthest parent link.

One of the biggest issues the team dealt with was querying the nested JSON element for a specific course. In order to improve the query speed, the team separated each of the merged JSON program files into the same file, giving each element an attribute of its program. The JSON-querying algorithm first returns the element for that program, followed by a recursive algorithm that searches through the nested courses to access the right match based upon the name attribute passed through the search field parameter. To complete the process, the algorithm returned the element to the top of the recursive call stack.

This element now appears as the root of the tree with the first level children visible. The third degree from the node, the children of children, are initially collapsed and not visible. Drawing links between nodes required diagonal objects. The diagonal objects (d3.svg.diagonal()) are graphic elements which draw a cubic bezier curves between two specific points. This type of object is widely used in the D3 library, especially when needing to draw a connection between two different elements in the drawing area. Each node is represented with a small circle (SVG circle). The circle is drawn at (x,y) position, with the x and y values defined by the node object automatically calculated by the cluster.nodes() function.

Styling & Layout

The team decided to keep color encodings to a minimal throughout the project and to maintain a common color palette. In addition, the team chose to keep interactions at a minimal level in order to satisfy the goal of a simple, intuitive approach for exploring course sequences. The tool included a tooltip which displayed important course attributes such *course title*, *course description*, *credit count*, and *credit type* upon mouse over of a node.

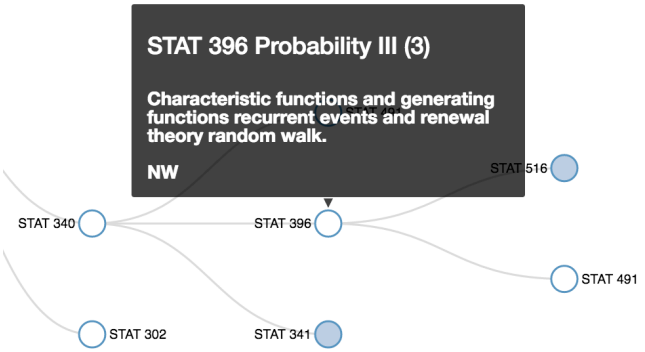

Figure 4: Tooltip showing course title, course description, credit count, and credit type.

The tooltip allowed for additional information in the case the user wanted to learn about the full description of the course. In addition, the tool implemented the open source Bootstrap framework for basic styling and page formatting.

RESULTS

The tool, now informally available to the University of Washington community, will improve the consumption of information of the vast course catalog. The primary use case of the tool is sequence exploration; the tool's usage will peak leading up to the few weeks before registration period. We've determined through user interviews that a

student will use this tool alongside the official course catalog. The two work well together, for they both provide similar, yet different features at the same time. For example, this visualization tool may not be the best way to casually browse a department's courses for the first time. However, this tool's value comes at a case when a user has a general idea for what they are looking for in the catalog in order to determine whether they are eligible to enroll in a particular course or not. The team will continue to maintain the project in the near future in order to ensure functionality and smooth usage for the community.

DISCUSSION

The goal of this tool was to help improve academic planning for students at the University of Washington. The team believes the project has definitely filled a void for a tool of this such. A primary reason for choosing this topic for the final project was due to the high amount of interest gathered from speaking with students during registration periods. Students have proclaimed their stress about the lack of a tool of this sort, for choosing courses to enroll can be a difficult task without the strict guidance of an academic advisor. The feedback received from this tool has been overwhelmingly positive, both from the audience during the final poster presentation on June 7th, 2016 and from one-on-one user interviews conducted. In fact, the team has received several recommendations to meet with the University of Washington Information Technology division (UW-IT) to suggest the implementation of a similar tool across the enterprise-wide UW system.

FUTURE WORK

The UW Prerequisite Course Explorer can be expanded in two major capacities: degree planning integration and user academic progress integration.

Degree Planning Integration

UW offers information for each department's admission and graduation requirements. The team could extend the tool to include this information in order to guide the user on the required courses needed for admission or graduation. Combining this information with prerequisites already visualized in the tool would provide a seamless academic planning experience.

User Academic Progress Integration

Implementing optional user input fields for courses previously enrolled will allow the tool to tailor a custom visualization specifically for the user. For example, the tool would hide the courses previously enrolled in to reduce clutter and focus on the courses a user would be interested in. Ideally, an open API from UW-IT allowing access to a user's transcript would be a great addition in order to immediately grab a student's academic progress, but this is unlikely to ever happen due to privacy reasons.

REFERENCES

1. Coursector CourseFocus. http://www.coursector.org/coursefocus/.
2. Data Driven Documents. https://d3js.org/.
3. Data Driven Documents: Collapsible Trees. https://bl.ocks.org/mbostock/4339083
4. UW Course Search. http://uwcoursesearch.com/.