A New Way to Shop

Building a Content-Based Recommendation System for Yale College Courses

Kira Tebbe

Senior thesis submitted to the
Department of Applied Mathematics
of Yale University

in partial fulfillment of the
requirements for the degree of
Bachelor of Arts

Adviser: Amin Karbasi

December 16, 2016

Table of Contents

Table of Figures & Tables

Section I: Introduction

Motivation

Yale students have flexibility in choosing their academic coursework: at the beginning of each semester, students have two weeks to sit-in various classes without commitment, calling “shopping,” before submitting their schedule. Students often have a few classes preselected that are required for either their major or distributional requirements. However, there are many options for the other classes students take in a semester, given that there are over 1,000 Yale courses and most take four or five. The process of selecting which courses you will attend during these first two weeks is called “blue booking,” referencing the original blue course offering books that students would leaf through before the semester began.

The methods that Yale students employ when blue booking varies by person. Some use online databases to sort all classes by difficulty and ranking, some take courses that are prerequisites for future classes they would like to take, some search through some department’s entire offerings. I sought to offer a novel way of searching for courses by creating a recommendation system based on course topics for the spring 2017 course offerings. The recommendation system, which will be described in length in the rest of the report, connects courses deemed similar by the keywords present in their descriptions. Students using the recommendation system start from a course they know they like and the recommendation system suggests similar courses.

Context

There have been many network science analyses on what is called the “*Wikispeedia* game” (West & Leskovec, 2012), which serves as inspiration for part of the methodology of this thesis. *Wikispeedia* is a game in which players are assigned two random Wikipedia pages and must click on hyperlinks to get from one to the other. An analysis of these human-driven paths is often compared to the shortest actual path of the network, which has also been calculated. However, Wikipedia relies on many users to submit and edit topic descriptions and hyperlinks, thus ensuring user-generated connections. Deciding how to connect Yale courses is not as predetermined, but the methods used will be described below.

A recommendation system can be seen as a modification of this Wikispeedia game. The user continues to explore the network, iteratively selecting classes that they prefer, as if searching for their ideal class as a target. Analyzing how users play this game, regardless of how the courses are connected, can give further insight in what mental schematic people employ when thinking about courses.

Section II: Data Collection & Preparation

API

The Yale course offerings for the spring 2017 term were pulled using the Yale Developer Portal (Yale University Developer Portal, n.d.). Both the "Course Subjects" and "Courses" APIs were used. The "Course Subjects" API was used to collect the abbreviations of all 138 departments at Yale. These abbreviations (e.g. AFAM) were used to pull all spring 2017 offerings from the "Courses" API (where "subject code" is a required input term). These returned 138 different JSON files. Some of these files were empty, meaning no undergraduate courses were being offered in the department.

The JSON files were consolidated into a single data frame, with each row corresponding to a single course. Careful attention was paid to recognizing cross-listed courses to avoid duplicating them in the data frame. The final data frame included one of the course numbers (e.g. AFAM060), the course title and description, and all other cross listings.

Figure 1

Dept1	Title	Description	Dept2	Dept3	Dept4
AFST340	Africa in the Era of the Slave Trade	Examination of the tumultuous changes experien...	HIST340	*NA*	*NA*
AFST449	Challenges to Realism in Contemporary African Fiction	Introduction to experimental African novels that c...	ENGL449	*NA*	*NA*
AFST413	Governance in Africa	International donor agencies, along with global a...	GLBL328	PLSC413	*NA*
AFST432	Development and Democracy in Africa	Introduction to development challenges in Africa. ...	PLSC414	*NA*	*NA*
AFST221	Egyptomania	Conceptual underpinnings of the use of ancient E...	ARCG221	HSAR234	NELC120
AFST435	West African Dance: Traditional to Contemporary	A practical and theoretical study of the traditional...	THST335	*NA*	*NA*
AKKD120	Elementary Akkadian II	Continuation of AKKD 110.	*NA*	*NA*	*NA*
AMST235	Language, Disability, Fiction	Portrayals of cognitive and linguistic impairment i...	ENGL354	*NA*	*NA*
AMST257	Modern Apocalyptic Narratives	The persistent impulse in Western culture to imag...	ENGL325	*NA*	*NA*
AMST348	Space, Place, and Landscape	Survey of core concepts in cultural geography and...	*NA*	*NA*	*NA*
AMST385	Trauma in American Film and Television	Origins, multiple meanings, and influence of the ...	*NA*	*NA*	*NA*
AMST441	Indians and the Spanish Borderlands	The experiences of Native Americans during cent...	HIST130J	ER&M370	*NA*
AMST466	Contemporary Historical Novels	Attempts of contemporary American authors to p...	ENGL444	*NA*	*NA*
AMST311	Latina/o New Haven	Introduction to the field of Latina/o studies, with ...	ER&M311	*NA*	*NA*

Department Affiliation

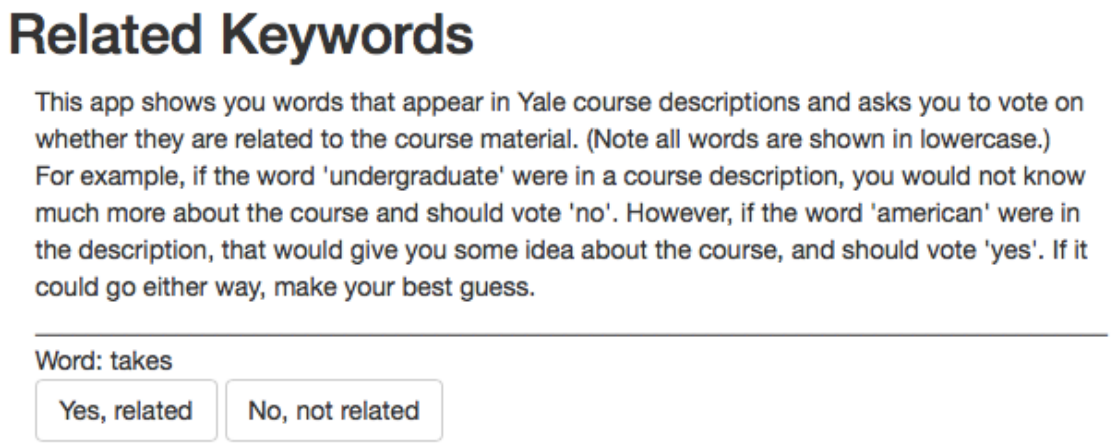
A preliminary network was constructed using departmental affiliation as a proxy for course topic similarity. Each course was connected to all other courses in its department, including any cross-listed departments. Each department was a clique, with the cross-listed courses acting as bridging ties between departments. However, this proxy for measuring course similarity was abandoned for practical reasons. Part of the motivation for this project was to provide students with a different method to find courses to shop. When searching for courses to take, it is practically easy to search by department, whether by a search with the department code on the Online Couse Information database or by finding the courses offered on that department's website. If the similarity network, on which the course recommendation system would be based, were rooted in department affiliation, recommended courses would likely remain in the department of the seed course, which would be of less benefit to students.

Keyword Similarity

After reading about the Wikispeedia game, it was decided that the course descriptions were to act as the measure of similarity. All words from course description were pulled for examination. After removing special HTML tags (e.g. “<p>”) and all other punctuation, the most common words were revealed to be “stop words,” such as “the” and “a.” A list of stop words was imported and removed from all of the description words (Stopwords, n.d.). The next most common words still did not contribute to the heart of the course topic, such as “students,” “material,” and “include.” As there were over 6,651 total unique words, a manual review of all words was not possible. This work was “outsourced” using a small Shiny application (Related Keywords, n.d.).

Shiny is a web application framework built on R, a popular function language (Shiny, n.d.). Shiny allows a R program to be hosted on a web browser, allowing remote access by users who do not need to have R installed on their own computers. The application welcomed users with a brief description of the purpose, a word, and two buttons (“yes, related” and “no, not related”). The description explained that a word should be deemed “yes, related” if it contributed to an understanding of what the course is about. It gives as example in the header to help inform the user as to the purpose of the game.

Figure 2



Of the 6,651 unique non-stop words used across the course descriptions, 1,052 words appeared in at least 5 course descriptions. The application would show a random one of these 1,052 words and a new word would appear after voting on the previous word. This process would continue indefinitely until the user decided to stop playing. The button clicks were recorded through connected to a MySQL database hosted on Amazon Web Services (AWS, n.d.). Once the application was prepared, it was launched and advertised to the author’s close friends. The Shiny application was left open for 14 days. In this time, there were 1,330 votes on 889 words. It was decided that words were to be removed if they had been voted on negatively during the game.

Senior Courses

An initial exploration of the course similarity matrix revealed some pairs with very similar keywords, most of which fell into a category of senior-only courses or courses defined by independent work. Examples of these courses are “The Senior Essay,” “Individual Study,” and “Individual Reading and Research.” Since the eventual goal of the recommendation system is to help students with their Yale course shopping experience and expose them to courses they may like but not otherwise have found. These senior-only and independent classes are generally decided beforehand: if students plan on taking one, they know. Thus, these are not classes that many students shop, since they are offered to such a selective group of students and often require faculty approval. Because of this rationale, these courses were removed from the database of courses. Removal was done by collected all of the similar words in these course titles, such as “senior” or “study,” and removing a course if it included two or more of these words in its title. In the end, 156 courses that fell into this category were removed.

Section III: Network Construction

Jaccard Index

The Jaccard index was used to serve as a proxy for similarity. Once the descriptions were converted into a "bag of words" and the irrelevant words were removed, similarity between two courses was measures by their number of overlapping keywords divided by the number of unique words in both course's descriptions (Hamers et al., 1989):

Figure 3

$$J(A,B) = \frac{A \cap B}{A \cup B}$$

This metric is useful for many reasons, one of which is its accounting for the differences in course description word lengths. The Jaccard index was used to conduct a similarity score between all pairs of classes. The result was a matrix where *m[i,j]* is the similarity score between courses *i* and *j*. This similarity matrix was used to determine which courses would be connected by an edge in the network.

Setting the Cutoff

The distribution of similarity scores across all pairs of courses is severely right skewed, with the majority of similarity scores smaller than 0.01. A "cutoff" score was needed to decide which pairs of courses would be connected in the base network.

To rigorously choose the optimal cutoff score, fifty different networks were constructed, each with a cutoff score incrementally (step size of 0.01) between 0.01 and 0.5, and three network attributes were measured: number of isolated nodes, the max degree, and the mean degree (see Figures 1, 2, & 3).

Figure 4 Max Degree by Cutoff Score

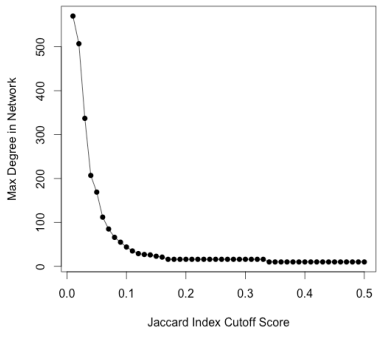


Figure 5 Mean Degree by Cutoff Score

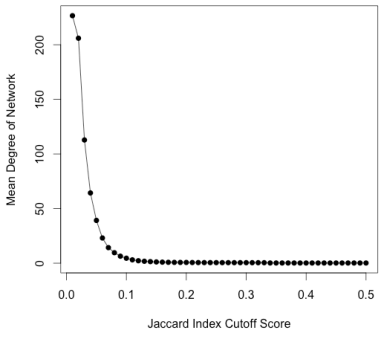
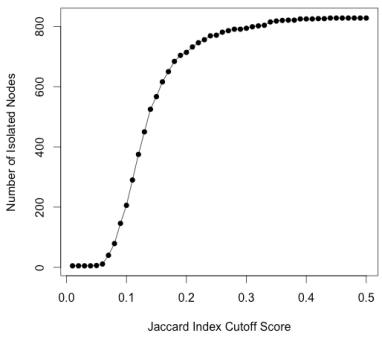


Figure 6 Number of Isolated Nodes by Cutoff Score



As one would expect, as the cutoff score rose, requiring courses to be more similar to be connected, the number of isolated nodes also rose. The plot, however, was not linear; rather it looked like a logistic growth plot. The mean degree decreased rapidly, from 247.6 to 5.6 as the cutoff decreased from 0.01 to 0.10, respectively. After 0.10, it leveled out, resembling an exponential decay plot. The maximum degree followed a similar pattern, with a sharp decrease in the first ten iterations, then stabilizing to around 17 after a cutoff of 0.37.

There seemed to be an ideal choice in cutoffs between 0.05 and 0.08, so the same table was created for values in that range with a step size of 0.001 (see Figures 7, 8, 9). The plots of values were not as drastic, with all three nearly linear. Eventually, a cutoff of 0.08 was chosen. This resulted in 79 isolated nodes, a maximum degree of 66, and a mean degree of 9.59 (see Figure 10).

Figure 7 **Mean Degree by Cutoff Score**

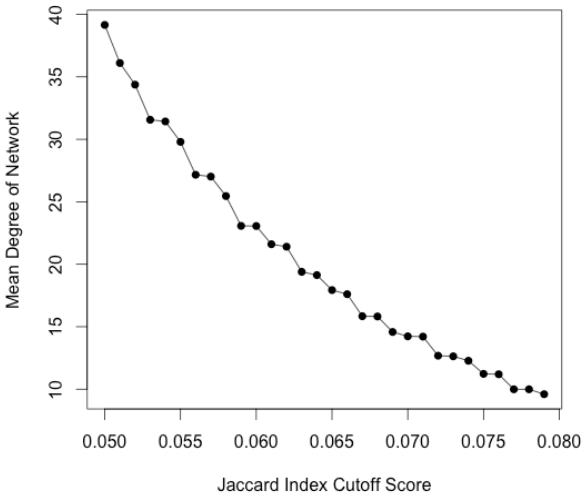


Figure 8 **Max Degree by Cutoff Score**

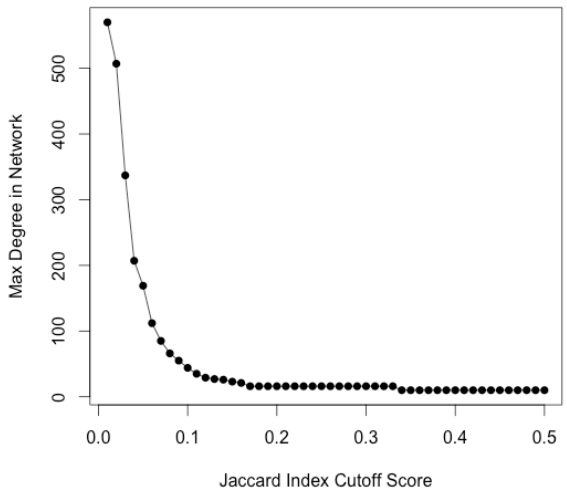


Figure 9 **Number of Isolated Nodes by Cutoff Score**

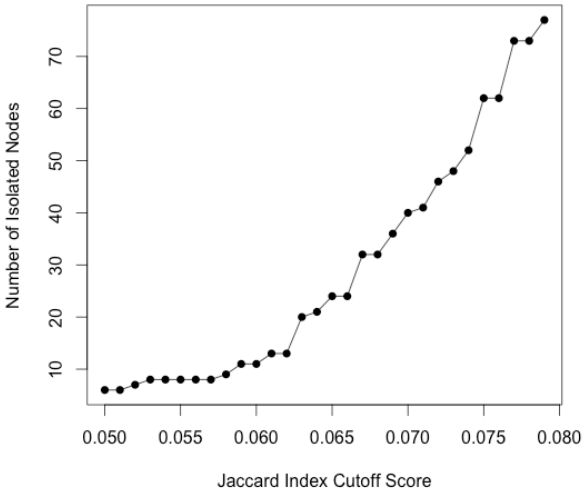
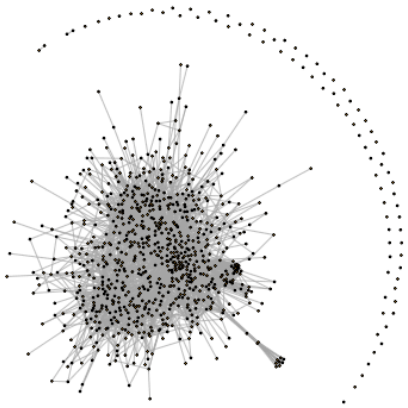


Figure 10 **Network Before Connecting Low Degree Nodes**



Language Classes

It was noticed that many of the courses with high degree in this base network were language courses, where keywords consisted only of "continuation," a department code, and a course number. In fact, 45 offered courses have those exact keywords (for a variety of departments). These courses are not ideal for the recommendation system, as they are a part of a series that a student could not take without completing the first course in the

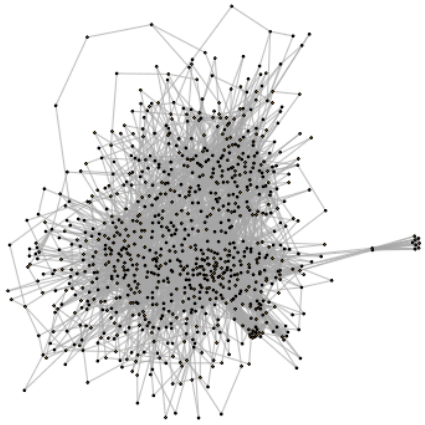
series in the previous semester. For this reason, these language classes were removed from the database of courses.

Adding & Removing Edges

The cutoff resulted in 79 isolated nodes and 68 nodes with degree 1. In order for users to search the network, it needs to be both connected and have no "leaves" (nodes with a degree of 1). If a user is searching the network and comes across a course that is only connected to one other, there would be nowhere to go except back to the previous course. Thus, an overall minimum degree of 2 was desired. This was achieved by finding all courses with a degree of 0 or 1 and adding edges between it and other courses (with the highest similarity scores) until it had a degree of 2. Caution was taken to ensure the added edges did not already exist in the graph, as such multiplexity occurred when the algorithm was implemented initially.

Figure 11

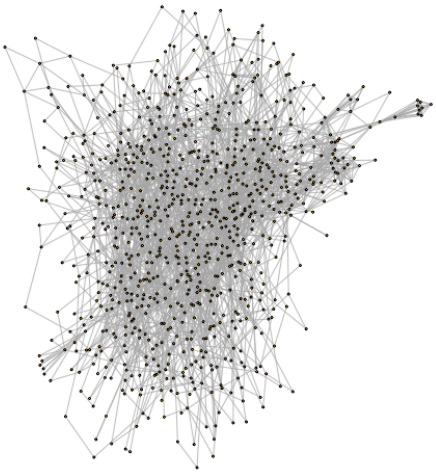
Network After Connecting Low Degree Nodes



This final network (seen in Figure 12) had a variety of degrees, even though the minimum was made to be 2. The maximum degree in the network, as is, was 67. Since the game would require a fixed number of buttons, there needed to be a maximum degree across the entire network so that all neighbors would be able to be shown to the user. To circumvent this logistical challenge, edges were removed from the network until the maximum degree was 11. With this, users would be presented anywhere from 2 to 10 new options, which is much easier to read and choose from than over 50.

Figure 12

Network After Removing Excess Edges



To remove these edges, all of the courses with a degree larger than 11 were identified. For each of these courses, a seed was set, so the algorithm was deterministic, and one of the edges was sampled. The network with that edge removed was tested for keeping an overall minimum degree of 2. If the degree decreases, that edge was kept and a new one was sampled. If the degree stayed constant, the edge was permanently removed. This process, of selecting an edge and checking the connectivity of the network without it, was iterated across all classes until none had degree larger than 11. (It is important to note that this algorithm does not

involve the original similarity indices. It samples all edges with equal probability.)

The Three Different Networks

The process above created a preliminary network that will henceforth be referred to as the "base" network. To find the most efficient and searchable network, and to test for those qualities, two additional networks were created. The base network acts as the experimental control. The second network adds 276 edges to the base network, but does so strategically. The third network is the same base network with 276 randomly added edges.

Making the Smart Network

The number of edges to add was determined by the algorithm used to strategically add edges. In this "smart" network, an edge was added between two random vertices with a probability proportional to the distance between the two vertices. There was also a normalization factor based on the total distance from the source node to all other nodes in the network. The normalization factor for u and the probability of adding an edge from u → v were calculated as the following:

Figure 13

$$N_u = \sum_v d^{-2}(u, v)$$

$$\Pr(u \to v) = \frac{d^{-2}(u, v)}{N_u}$$

Checks were put into place to ensure that the two vertices were not already connected. Additionally, if u or v had degree 11, a random edge was removed from it since the maximum degree needed to remain at 11, so adding another edge could only occur if one were removed. When an edge was removed, it was ensured that the vertex at the other end of the edge did not have degree 2 (since removing that edge would make it degree 1). If an edge needed to be removed, because u or v had degree 11, the network with the randomly selected edge removed was tested for a minimum degree of 2. If the minimum degree was still 2, the edge was removed from the real network. If not, it continued to a different edge and the same checks were made until an edge was successfully removed.

The algorithm between all pairs of classes took some time to run and integrity checks were run on the resulting network to ensure it stayed connected, the minimum degree was 2, and the maximum degree was 11. There were 286 added edges.

Making the Random Network

To validate the methodology of the "smart" network, a third network was created that had the same number of edges added as the "smart" network, but randomly instead. This "random" network serves as a check that any benefit brought about by the additional edges in the "smart" network is a result of the methodology and not just the presence of more edges. The random edges were added by first randomly sampling two nodes u and v. If they were connected, it would randomly sample again. If not, it would add an edge between u and v. (If u

or v had degree 11 before adding the edge, and random edge was removed, similar to what occurred in making the “smart” network.) This was repeated until the number of successfully added edges equals the number of edges added to the “smart” network. It is important to note that removing an edge from a vertex with degree 11 so that a new edge may be added keeps the number of total edges constant.

Traits of the Three Networks

The most notable difference between the base graph and the smart and random graph is the distribution of degrees. While all are between 2 and 11, the base graph has a much higher proportion of degree 2 nodes than the random and smart network. The smart network intentionally connected nodes that were more removed from the rest of the graph, so it seems logical that the added edges made the proportion of nodes with degree 2 much lower. The random network also has this property, though this is more from chance.

More differences between the networks can be seen when looking at the mean path length, diameter (longest path in the network), and mean degree. The random and smart networks have similar mean path lengths, which are shorter than the base network. They also have a shorter diameter than the base network, although it was hypothesized that the smart network would have a smaller diameter than the random network because of how the extra edges were added. The mean degrees of the random and smart network are the same since they added the same number edges to the base graph.

Figure 14

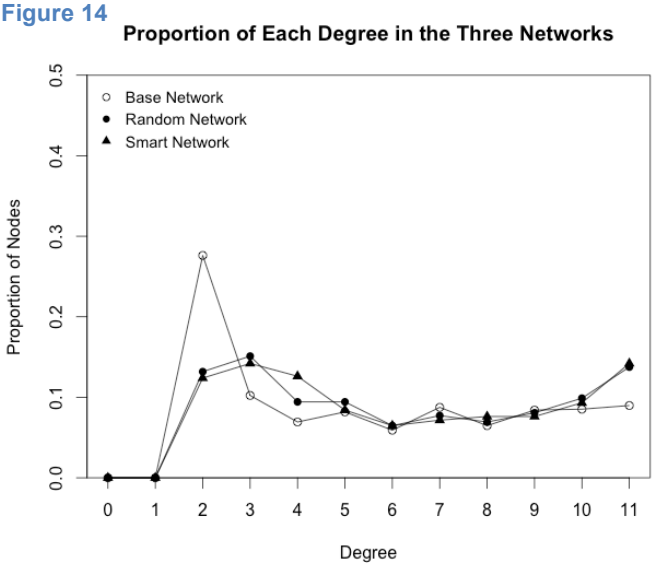


Figure 15

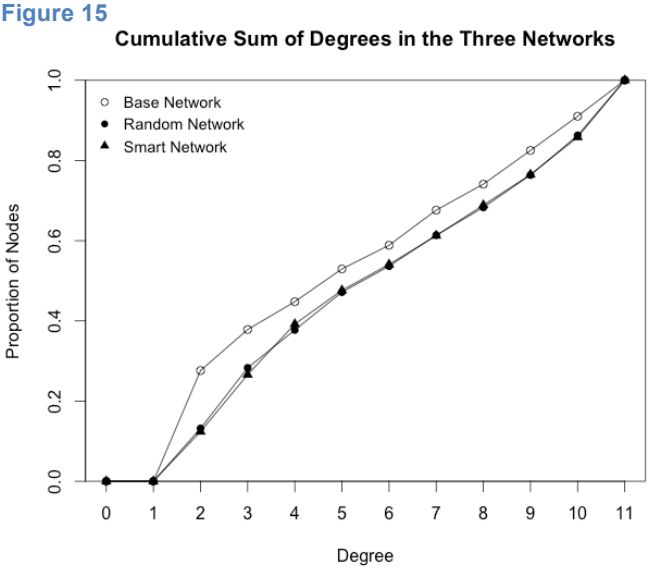


Table 1

	Mean Path Length	Diameter	Mean Degree
Base Network	4.52	10	5.63
Random Network	4.11	7	6.28
Smart Network	4.14	8	6.28

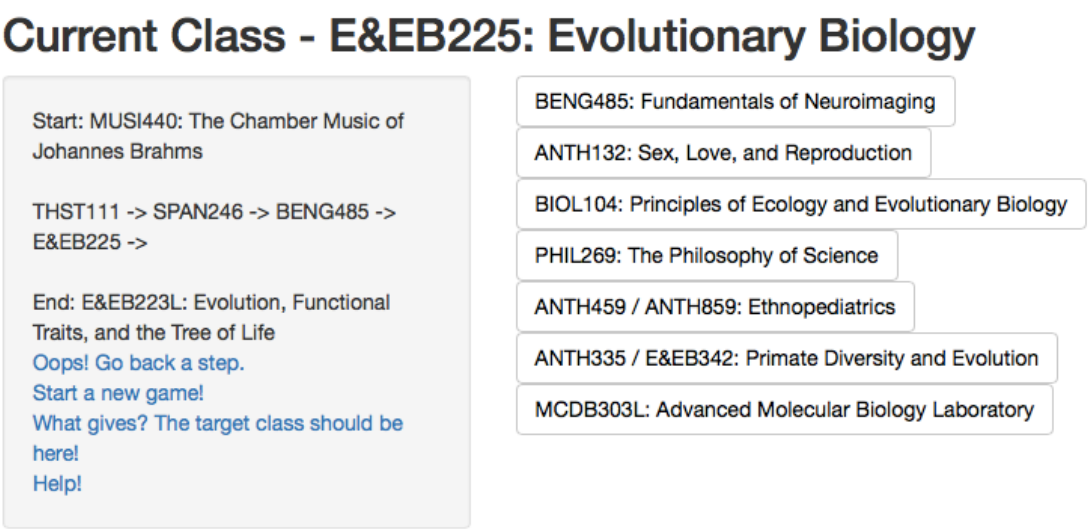
Section IV: Game Deployment

General Layout

To test the navigability or the three networks, three versions of the same Shiny application were deployed (Base Network, n.d.; Random Network, n.d.; Smart Network, n.d.). Users playing a game did not know which network was associated with which network because they all looked identical.

There are three main sections to the layout of the game: the header, sidebar, and main panel. The header updates on every click to remind users which course they are currently "at," to know which course's neighbors they are seeing. The main panel presents these neighbors as horizontal buttons stacked on top of each other. The buttons, like the header, present the course department(s), number(s), and title.

Figure 16



The sidebar contains a variety of information. At a game's initialization, the user is faced with their "start" and "end" courses. These two courses are randomly selected at the beginning of the game from the list of all courses. These courses are presented with the same full listing as the main panel buttons and header. As the game is played, new buttons appear. In the space between the start and end course names is the path that the user has traveled on so far. In an effort to avoid clutter and make the information easily referenced, the path is summarized with just the main department code and number. There is also a "back" button to make the game more accommodating for user mistakes and to mimic what is available in the *Wikispeedia* game. There is also a button to start a new game and to suggest a connection.

If the user believes that their currently selected class is very similar to the target class, selecting the button would suggest that those two courses should be connected. This does not change the network during gameplay, but can be used

as another measure of the success of the strategically added edges (beyond just the number of steps).

To facilitate an understanding of the game, a “help” button was also present in the sidebar. It explains the goal of the game and the purpose of the “Back” and “Reset session” buttons. It also explains the “Suggest an edge” button. It does not, however, mention how the network of courses was created. This was intentionally omitted as to not influence the strategies used by participants. It is hypothesized that people use a variety of methods, such as selected courses with the same department affiliation, similar number (e.g. a 200-level course), or by topic like in the *Wikispeedia* game (West & Leskovec, 2012).

To ensure that the user knew when they had found the target course, and to avoid any human-based oversight enhanced by exasperation, the target course was colored blue if it were presented as an option. When the user successfully wins the game, the main panel with the buttons is replaced with a winning declaration, inviting them to play again.

Connection to the Database

To capture how the users play the game, each of the three versions are connected to a MySQL database. Each network was associated with its own table in the database. Any interaction that the user had with the game is recorded in the table, which has that following six fields: session ID number, source class, target class, the chosen class (at each step), going back a step (with the back button), and suggesting a link. The session ID number was created by sampling an integer between 1 and 1,000,000 at the beginning of a session. While it is possible that two different sessions have the same session ID (although it is likely given that less than 1,000 games will be played), this will not interfere with analysis since there is a check for very discontinuous gap between session IDs in the database.

Section V: Analysis of Game Results

All Search Results

Overall, there was a low completion rate of each search across all three networks (see Figure 17). The smart network had the highest completion rate, 36%, followed by the random network, 26%, and the base network, 24%. This metric suggests that the smart network is the most navigable. It also suggests that users either were impatient or became frustrated when playing.

Figure 17 Completion Rates For Each Network

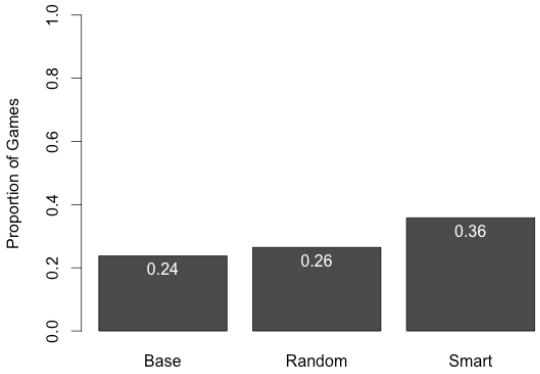
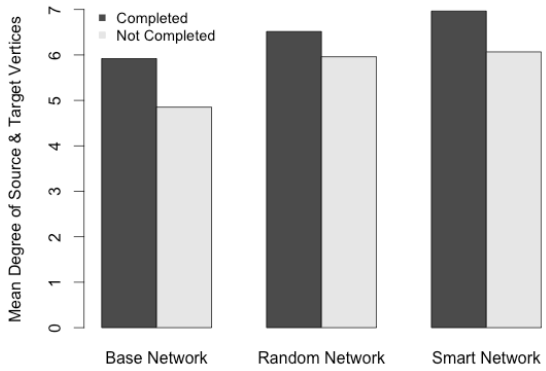


Figure 18 Game Completion by Mean Degree for Each Network



Across all three networks, searches that were from or to high degree nodes were more likely to be completed (see Figure 18). This makes logical sense, as trying to find a node with very few edges means a user would have fewer avenues by which to find it.

Across all networks, searches that were eventually completed were, on average, shorter than those that were uncompleted (see Table 2). This suggests user consistency across games - users are more likely to finish a search that is easier. We also see from this table that the average path length in the base network is longer than the average path length in the random and smart network, which is intuitive since the latter two networks have additional edges.

Table 2

Average Path Length of Searches		
	Completed Searches	**Uncompleted Searches**
Base Network	4.16	4.83
Random Network	3.66	4.20
Smart Network	3.79	4.33

To look at more differences between completed and uncompleted searches, the average traveled path length was examined (see Table 3). In both the base and the random network, completed searches often involved a longer traveled path than those that were uncompleted. It was hypothesized that people would click endlessly and eventually give up, but on average those who completed the game were the ones who did more searching. The mean traveled path across all networks is also larger for completed searches (15.60) than uncompleted searches (13.80). It is not clear why the opposite trend appears in the smart network.

Table 3

Average Traveled Path Length		
	Completed Searches	**Uncompleted Searches**
Base Network	14.37	12.44
Random Network	17.88	13.78
Smart Network	13.90	15.43

Completed Search Results

Table 4

Mean Difference Between User Path & Shortest Path		
	Paths Without Cycles	**Paths With Cycles**
Base Network	6.32	10.21
Random Network	6.31	10.22
Smart Network	7.14	14.10

Table 4 presents two additional metrics by which to measure network success. The first column measures the average difference (for each of the three networks) between the user's cycle-free path and the actual shortest path. Unexpectedly, the smart network has the largest difference, meaning users who complete a search do so with an average of one additional step than those in the base or random network.

Investigations were also made to the betweenness centrality of each network and of the source and target nodes in both completed and uncompleted searches (see Table 5). Betweenness centrality was chosen specifically for its importance in traversing between different clusters in the network. A high betweenness centrality could serve as a complimentary metric to degree of the source and target nodes. Of all searches, the smart network had the highest betweenness centrality of its source and target nodes. This pattern is mirrored in the trend in the uncompleted searches. All of the source and target nodes in those searches had around average or below average betweenness centrality.

Notice the pattern in the completed searches. Both the random and smart networks show below average betweenness centrality in the completed searches. This might imply that source and target betweenness centrality is not the only factor contributing to a successful search; that there are other network attributes that help the user find the target. However, the average betweenness centrality for completed searches in the base network is significantly above average. This suggests that only searches that were able to be completed in the base network were the ones that had source and target nodes with an unusually high betweenness centrality.

Table 5

Betweenness Centrality of Source & Target Nodes			
	All Searches	Completed Searches	Uncompleted Searches
Base Network	1282.20	1784.30	1120.50
Random Network	1413.68	1337.36	1441.21
Smart Network	1542.47	1471.79	1584.23

Comparison to *Wikispeedia* Results

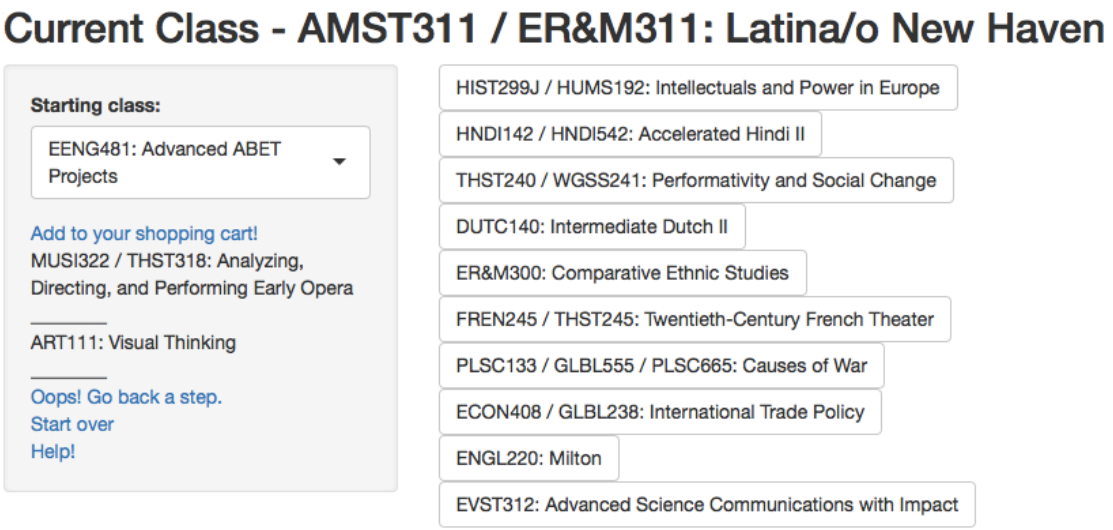
The findings from these networks are different from those in West et al. They found an incompletion rate of 54%, while the same rates for the three versions ranged from 23% to 36%. They also found that the "median human game path is only one click longer than median optimal solution" (West & Leskovec, 2012:620). Of the completed searches in the three networks, the median game path, excluding cycles, is 6.61 steps longer than the median shortest path.

The differences in findings could be due to a few fundamental differences between this game and *Wikispeedia*. The largest difference is the heightened familiarity that users have with Wikipedia and what hyperlinks to expect. West et al. conjecture that “even without knowing the set of all existing links, the Wikipedia graph is efficiently navigable for humans because they have an intuition about what links to expect" (West & Leskovec, 2012:622). The same cannot be said for the underlying networks presented here. Students do have a general familiarity of the departments at Yale, but not to all courses and their descriptions. Importantly, Wikipedia pages allow you to go to “hubs,” macro-level topics that connect to many other pages (e.g. “water,” “Europe). While a new course option may take you to a new department, there are no purely macro-level courses.

Section VI: The Recommendation System

After searches on the three networks were conducted and analyzed, it came time to build the recommendation system. It was decided that the smart network would serve as the foundation of the recommendation system given the results of the analysis in Section V. As one could predict, the recommendation system came in the form of a Shiny application, with small modifications made from the applications exploring the networks (CourseRec, n.d.).

Figure 19



The header and main panel of the application remain the same as before. There are a few additions to the sidebar. The user is presented with a drop-down selector tool to act as the starting node. (A random course is fill when the app is initialized.) The user can type in the department code, number, or words in the course title, and the selector tool will limit to what they have typed. By selecting a new starting course, the application shows a new set of neighbors. Exploration of the network is the same as from the previous application. When a user comes across that they particularly like, they can click the sidebar button “Add to your shopping cart!” This adds the course department code(s), number(s), and title to the sidebar for their reference. Users are encouraged in the “About” tab to copy or take a picture of these classes, as they do not persist and are inaccessible after the users exists the application.

Section VII: Limitations & Future Enhancements

There are various possible future additions and enhancements to this project. One simple way to make to project better would be to get more plays of each game, giving more data on the ways user interact with each network. The base network, random network, and smart network had 80, 121, and 81 plays, respectively. The author promoted the games to her friends via social media, but a larger dissemination of the games could have yielded more accurate results. (The reason behind the difference in plays of each game version is unknown.)

Often students take a class not only for the content, but also for its status as a prerequisite for future classes they want to take. Some initial ideas for the project involved making a directed network based on what courses are prerequisites for other courses. However, it is generally STEM courses that have prerequisites; many humanities and social science courses are accessible by application, preference to the major, or most uncategorized methods determined by the professor. This idea was not pursued because of this ambiguous foundation.

If the recommendation system is meant to aid in Yale College student's selection of courses, it would be beneficial to be able to filter the recommended courses by date and time. For example, perhaps a student wants to explore classes to fill a specific vacancy in their schedule; the recommendation system could simply only present courses that fulfill these restrictions. Such an extension would require incorporating dates and times into the courses data frame, as it is currently not recorded. There would be additional problems of the restricted network being possibly disconnected, especially of the restrictions placed upon it are very thorough.

All three versions of the Shiny game included an option to "suggest a connection" if the user thought that the present course should be connected to the target course. The smart network might have benefited by incorporating these suggestions to more subjectivity. This opportunity was not explored, unfortunately, due to time restrictions.

Although exclusively using a network based on department affiliation was abandoned, using a mixture of the description keyword similarity index and department affiliation could enhance the recommendation system. It would need to be determined how much weight to assign to the two different metrics (keyword versus department similarity), but it seems plausible that finding an optimal balance could be beneficial. Such a balance could perhaps be found through modeling.

By constructing a similarity index based on description keywords, this project crafted a "content-based filtering" recommendation system (Pazzani & Billsus, 2007). However, there exists another type of recommendation system that is "collaborative filtering" (Schafer, Frankowski, Herlocker, & Sen, 2007). Exemplified by Amazon product recommendations, collaborative filtering gathers many users' past behavior and predicts their future preferences by looking at the preferences of similar users. Collaborative filtering is behind the product recommendations introduced as "customers who bought (item 1) also bought

(item 2).” A famous example of this is the Target pregnancy marketing scandal, when they began to send coupons for pregnancy-related items to a teenage girl, to the aghast of her father (Hill, 2012). He called the company outraged, but apologized a few weeks later when he learned that his daughter was, in fact, pregnant. Target identified this before the girl’s own father by noticing a similar pattern of items bought on a similar timeline of other users who had eventually signed up for a baby registry. For this project, it seemed less feasible to make a course recommendation system based in collaborative filtering, as that would require the past course history of many students, which is confidential data that is not easily attained. However, if such data were available, the recommendation system could provide more holistic suggestions. There might be patterns among selection of classes that are not topical, unlike on what the current recommendation system is constructed.

Appendix

This project was completed entirely in R, using the *rjson, shiny*, *igraph*, *RMySQL*, and *pool* packages. What follows is a description of the files and algorithms used during the project.

scraping.Rmd

This file first references the two APIs to gather the course department codes and the courses within each department. It makes a JSON file for each department's courses:

```
system('curl -X GET -v "https://gw.its.yale.edu/soa-
gateway/course/webservice/subjects?apiKey=[hidden]&termcode=201701" |
python -m json.tool > courses.json')

depts<-unlist(fromJSON(file='courses.json'))

baseCommand <- 'curl -X GET -v "https://gw.its.yale.edu/soa-
gateway/course/webservice/index?apiKey=[hidden]&termcode=201701&subjectC
ode="'
endCommand <- ' | python -m json.tool > '

for(i in seq(from=1,by=2,length.out=length(depts)/2)){
  fileName<-paste(gsub("&","",depts[i]),".json",sep="")
  depts[i] <- gsub("&", "%26", depts[i], fixed=T)
  systemCommand<-paste(baseCommand,depts[i],endCommand,fileName,sep="")
  system(systemCommand)
}
```

It then reads in all of the courses from each JSON files and converts the data to a manageable data frame, consolidated across all cross-listed courses (see Figure 1):

```
jsonFiles <- list.files()

## making the course data frame
courses <- as.data.frame(matrix(NA, nrow=10, ncol=12))
colnames(courses) <- c("Dept1", "Title", "Description",
                       "Dept2", "Dept3",
                       "Dept4", "Dept5", "Dept6",
                       "Dept7", "Dept8", "Dept9", "Dept10")
courseLoop <- 0
for(i in 1:length(jsonFiles)){
  if(file.size(jsonFiles[i])==0) next #skipping empty files (0 bytes)
  file <- fromJSON(file=jsonFiles[i])
  if(length(file)==0) next # skipping empty files (3 bytes)
  for(j in 1:length(file)){
    # checking to see if course is new to data frame
    if(length(which(file[[j]]$subjectNumber == courses$Dept1 |
                    file[[j]]$subjectNumber == courses$Dept2 |
                    file[[j]]$subjectNumber == courses$Dept3 |
                    file[[j]]$subjectNumber == courses$Dept4 |
                    file[[j]]$subjectNumber == courses$Dept5 |
                    file[[j]]$subjectNumber == courses$Dept6 |
                    file[[j]]$subjectNumber == courses$Dept7 |
                    file[[j]]$subjectNumber == courses$Dept8 |
                    file[[j]]$subjectNumber == courses$Dept9 |
                    file[[j]]$subjectNumber == courses$Dept10 )) == 0){
        courseLoop <- courseLoop + 1
        courses[courseLoop, 1:3] <- c(file[[j]]$subjectNumber,
                                      file[[j]]$courseTitle,
                                      file[[j]]$description)
        ## adding cross-listed if need be
        if(length(file[[j]]$scndXLst)>0){
          col <- min(which(is.na(courses[courseLoop,]))) #first NA in row
          for(cross in 1:length(file[[j]]$scndXLst)){
            courses[courseLoop,col+cross-1] <- file[[j]]$scndXLst[cross]
          }
        }
        ## add primXLst if not in row
        if(!is.null(file[[j]]$primXLst)){
          col <- min(which(is.na(courses[courseLoop,]))) #first NA in row
          if(!(file[[j]]$primXLst %in% courses[courseLoop,])){
            courses[courseLoop,col] <- file[[j]]$primXLst
          }
        }
    }
  }
}
```

similarity network.Rmd

This first removes all “senior-only” courses, determined by the presence of two keywords in the title. It also lists all of the most common words across all descriptions (excluding generic stop words), which became the content for the Shiny voting application (see Figure 2). It then creates the keyword list for each course, after removing the stop words and negatively voted words:

```
stopWords<- read.csv("stopwords.csv", as.is=T)
toRemove <- read.csv("wordsToRemove.csv", as.is=T)
toRemove <- toRemove[,3]
toRemove <- unique(toRemove)

keepKeywords <- function(descrip){
  words <- strsplit(descrip, " ")
  words <- unlist(words,recursive=FALSE) # make one big list
  for(i in 1:length(words)){
    #splitting and removing a weird string
    if(length(grep(" ",words[i])) > 0){
      words <- c(words,gsub( " .*$", "", words[i]))
      words <- c(words,gsub( ".* ", "", words[i]))
      words <- words[-i] #string with character
    }
  }
  words <- gsub("\\<[^\\]]*\\>", "", words, perl=TRUE) # removing between < >
  words <- gsub("\n","",words) # removing \n notation
  words <- gsub("[[:punct:]]","",words) # removing punctuation
  words <- tolower(words) #make lower case
  # removing stopwords
  for(i in 1:nrow(stopWords)) {
    rows <- which(words == stopWords[i,1])
    if(length(rows) == 0) next
    words <- words[-rows]
  }
  ## removing confimed bad words
  for(i in 1:length(toRemove)) {
    rows <- which(words == toRemove[i])
    if(length(rows) == 0) next
    words <- words[-rows]
  }
  words <- unique(words)
  return(paste(words, collapse = ' '))
}
```

The language classes that were exclusively continuations of previously courses were also removed. After some exploration of the keywords, a similarity matrix is constructed based on the Jaccard index of these newly created keyword lists:

```
n <- nrow(courses)
adj <- matrix (NA, nrow=n, ncol=n)
rownames(adj) <- courses$Dept1
colnames(adj) <- courses$Dept1
for(row in 1:(n-1)){
  rowList <- unlist(strsplit(courses$Keywords[row], " "))
  for(col in (row+1):n){
    if(row == col) next
     colList <- unlist(strsplit(courses$Keywords[col], " "))
     if((length(intersect(rowList, colList)) / length(union(rowList, colList))) == 1){
       adj[row, col] <- NA
       adj[col, row] <- NA
     } else{
       adj[row,col]<-length(intersect(rowList,colList))/length(union(rowList,colList))
       adj[col,row]<-length(intersect(rowList,colList))/length(union(rowList,colList))
     }
  }
}
```

The highest scores are examined, along with the within and between department scores (which end up not being used in further analysis). Various incremental cutoff scores are used to create and analyze network attributes, first between 0.0 to 0.5 (see Figures 4, 5, and 6) and then between 0.05 to 0.08 (see Figures 7, 8 and 9). It is from these plots that a final cutoff (0.08) was chosen.

making network.Rmd

This file first recreates the adjacency matrix used previously. It looks at the nodes that are isolated or only have a degree on one, and then uses the similarity score matrix to connect these nodes to their most similar neighbors (even if those scores are below the decided cutoff):

```
Adj <- adj >= 0.08
net <- graph_from_adjacency_matrix(Adj, mode="undirected")
lowDegree <- which(degree(net) <= 1)
for(i in 1:length(lowDegree)){
  vertexName <- names(lowDegree[i])
  scores <- adj[vertexName,]

  if(degree(net, v = vertexName) == 0){
    edges <- names(sort(scores, decreasing=T)[1:2])
    net <- add_edges(net, c(vertexName, edges[1]))
    net <- add_edges(net, c(vertexName, edges[2]))
  }

  if(degree(net, v = vertexName) == 1){
    ## make sure no duplicate edges! check neighbors names
    neighbor <- names(neighbors(net, v=vertexName))
    edges <- names(sort(scores, decreasing=T))
    if(edges[1] != neighbor){
      net <- add_edges(net, c(vertexName, edges[1]))
    }
    else {
      net <- add_edges(net, c(vertexName, edges[2]))
    }
  }
}
```

It plots the network before (see Figure 10) and after adding these extra edges (see Figure 11). It also removes excess edges from nodes with a large degree until the maximum overall degree becomes 11 (See Figure 12):

```
testSeed <- 1
largeDegree <- names(which(degree(net) > 10))
neighbors(net, v = largeDegree[99])

## looping through and removing edges & checking connectivity
for(i in 1:length(largeDegree)){
  class <- largeDegree[i]
  counter <- 0 ## counter to make sure no dead ends

  while(length(neighbors(net,v = class)) > 11){
    counter <- counter + 1
    if(counter > 68){
      print("Help! I'm stuck!")
    }
    testSeed <- testSeed + 1
    set.seed(testSeed)
    edgeName<-names(neighbors(net,v=class))[sample(length(neighbors(net,v=class)),1)]
    test <- delete_edges(net, paste(class,"|",edgeName, sep=""))

    ## checking connectivity & min degree
    if(is_connected(test) & min(degree(test)) == 2){
      ## ok to permanently remove edge
      net <- delete_edges(net, paste(class,"|",edgeName, sep=""))
    }
  }
}
```

adding by dist.Rmd

The normalization factor for each vertex was computed first by calculating the total distance from it to all other nodes in the graph (see Figure 13 for a formal definition):

```
library(igraph)
net <- read_graph("network.txt", format = "ncol", directed=FALSE)
N <- as.data.frame(matrix(NA, nrow = length(V(net)), ncol = 1))
rownames(N) <- V(net)$name
colnames(N) <- "Nu"

mat <- distances(net)
for(j in 1:length(V(net))){
  class <- V(net)$name[j]
  dists <- (1/mat[class,]) ** 2
  total <- sum(dists[is.finite(dists)])
  N[class,"Nu"] <- total
}
```

The normalization factors are then used in the algorithm that adds edges, which checks both for nodes with degree 11 (so an edge is removed before one is added) and check that the minimum degree remains at 2 (since randomly removing an edge could reduce the minimum degree). The algorithm added 276 edges:

```
mat <- distances(net) ** 2
set.seed(3)
for(i in 1:(length(V(net))-1)){
  class <- V(net)$name[i]
  Nu <- N[class, "Nu"]
  for(j in (i+1):(length(V(net)))){
    class2 <- V(net)$name[j]
    if(net[class, class2] != 0) { next } #skip if classes already connected
    dist <- mat[class, class2]
    ## add edge w probability equal to 1/(Nu*dist)
    if(as.logical(rbinom(1,size=1,prob=(1/(Nu*dist))))){
      degrees <- degree(net)
      ## check degree first
      if(degrees[class] == 11){
        #randomly remove an edge, making sure not to lower min degree
        NOTDONE <- TRUE
        while(NOTDONE){
          index <- sample(c(1:11), 1)
          test <- delete_edges(net, E(net)[from(class)][index])
          if(min(degree(test)) == 2){
            net <- delete_edges(net, E(net)[from(class)][index])
            NOTDONE <- FALSE
          }
        }
      }
```

```
if(degrees[class2] == 11){
        NOTDONE <- TRUE
        while(NOTDONE){
          index <- sample(c(1:11), 1)
          test <- delete_edges(net, E(net)[from(class2)][index])
          if(min(degree(test)) == 2){
            net <- delete_edges(net, E(net)[from(class2)][index])
            NOTDONE <- FALSE
          }
        }
      }
      ## add an edge!
      net <- add.edges(net, c(class, class2))
    }
  }
}

## integrity checks
min(degree(net)) == 2
max(degree(net)) <= 11
```

adding random.Rmd

This mirrors the “adding by dist.Rmd” code, but adds the 276 edges randomly using an algorithm described in Section III. It has various validity checks to ensure the correct number of edges was added and the desired network attributes (i.e. degrees between 2 and 11) were maintained.

```
edgesBefore <- gsize(net)
numToAdd <- 286
counter <- 0
set.seed(2)
while(counter < numToAdd){
  ## randomly sample two nodes
  nodes <- sample(V(net)$name, 2)
  ## proceed only if no edge already between
  if(net[nodes[1], nodes[2]] == 0){
    ## check degree -> if either = 11, remove random edge & replace w new one
    if(degree(net, v = nodes[1]) == 11){
      NOTDONE <- TRUE
      while(NOTDONE){
        index <- sample(c(1:11), 1)
        test <- delete_edges(net, E(net)[from(nodes[1])][index])
        if(min(degree(test)) == 2){
          net <- delete_edges(net, E(net)[from(nodes[1])][index])
          NOTDONE <- FALSE
          counter <- counter - 1
        }
      }
    }
    if(degree(net, v = nodes[2]) == 11){
      NOTDONE <- TRUE
      while(NOTDONE){
        index <- sample(c(1:11), 1)
        test <- delete_edges(net, E(net)[from(nodes[2])][index])
        if(min(degree(test)) == 2){
          net <- delete_edges(net, E(net)[from(nodes[2])][index])
          NOTDONE <- FALSE
          counter <- counter - 1
        }
      }
    }
    ## add an edge!
    net <- add.edges(net, c(nodes[1], nodes[2]))
    counter <- counter + 1
  }
}
edgesAfter <- gsize(net)

## validity check
edgesAfter - edgesBefore == numToAdd
min(degree(net)) == 2
max(degree(net)) <= 11
```

analyzing networks.Rmd

This looks at the different attributes of the three networks (base, smart, & random). There are two graphs that show the different proportion of each degree in the networks (see Figure 14 for proportion plot and Figure 15 for cumulative plot). The code that created non-cumulative graph is presented below:

```
library(igraph)
netSmart <- read_graph("networkDist.txt", format = "ncol", directed=FALSE)
cumsumValues <- function(net){rkRandom.txt", format = "ncol", directed=FALSE)
  a<-table(degree(net))network.txt", format = "ncol", directed=FALSE)
  a<-cumsum(a)
  a <- a / a[length(a)]se))/880
  return(a)degree(netSmart))/880
} <- table(degree(netRandom))/880
a<-cumsumValues(netBase)
b<-cumsumValues(netSmart)
c<-cumsumValues(netRandom)
     xlim =c(0,11),
     xaxt="n", ylab="Proportion of Nodes",
     main ="Proportion of Each Degree in the Three Networks",
     xlab="Degree")
axis(1,at = 0:11,0:11)
lines(x= 0:11,
      y = c(0,0,a))
points(c(0,0,b),x=0:11,pch=16)
lines(x= 0:11,
      y = c(0,0,b))
points(y=c(0,0,c),
       x= 0:11,pch=17)
lines(x= 0:11,
      y = c(0,0,c))
legend("topleft", c("Base Network", "Random Network", "Smart Network"),
       pch = c(1, 16, 17), cex=0.9, bty='n')
```

The code creating the cumulative plot is essentially the same, except the tables that are plotted are calculated in the following way:

reading DB.Rmd

This file did all of the analysis of the user's gameplay on each network. It first calls a function that reads in each network's corresponding database table and extracts the desired traits. Before calling that analysis function, two smaller functions were written to adjust each gameplay session by accounting for the "back" button and removing any cycles on which the user may have traveled:

```
removeBacks <- function(sub){
  ## if back == 1 -> remove it and row above
  while(length(which(sub$back == 1)) > 0){
    backs <- which(sub$back == 1)[1]
    sub <- sub[-((backs-1):backs),]
  }
  return(sub)
}
```

```
removeCycles <- function(sub){
  if(length(sub$choice[duplicated(sub$choice)]) > 0) {
    KEEPGOING <- TRUE
    index <- 1
    while(KEEPGOING){
      cycles <- sub$choice[duplicated(sub$choice)]
      if(sub$choice[index] %in% cycles){
        ## find rows with that cycle and slice from latest one
        last <- max(which(sub$choice == sub$choice[index]))
        sub <- sub[-seq(from = (index+1), to = last, by = 1),]
      }
      index <- index + 1
      if(index > nrow(sub)){
        KEEPGOING <- FALSE
      }
    }
  }
  return(sub)
}
```

This function consolidates each gameplay session into a single row with 11 attributes: the session ID, if the search was completed or not, the degree and name of the source class, the degree and name of the target class, the user's traveled path (including any cycles), the user's cycle-free path, the true shortest path in the network between that source and target class, how many times the user used the "back" button, and how many duplicated courses the user visited. It takes two inputs – the network and the entire table of gameplay (recorded in the database):

```
analyzeTable <- function(net, tab){
  df <- as.data.frame(matrix(NA, nrow = 1, ncol = 11))
  colnames(df) <- c("session_id", "completed",
                    "sourceDegree", "sourceName",
                    "targetDegree", "targetName",
                    "traveledPath", "cyclefree", "realPath",
                    "backs", "duplicates")
  shortestPaths <- shortest.paths(net, v=V(net), to=V(net))
  deg <- degree(net)
  idList <- unique(tab$session_id)

  for(i in 1:length(idList)){
    id <- idList[i]
    sub <- tab[which(tab$session_id == id), ]
    sub <- sub[sub$suggestion != 1,]
    ## fixing if they go back to first screen
    bad_ones <- sub$choice == ""
    sub$choice[bad_ones] <- sub$`source`[bad_ones]

    df[i, "session_id"] <- id
    df[i, "backs"] <- length(which(sub$back == 1))
    df[i, "duplicates"] <- length(unique(sub$choice[duplicated(sub$choice)]))
    df[i, "realPath"] <- shortestPaths[sub[1,"source"], sub[1,"target"]]
    df[i, "sourceDegree"] <- deg[sub[1,"source"]]
    df[i, "sourceName"] <- sub[1,"source"]
    df[i, "targetDegree"] <- deg[sub[1,"target"]]
    df[i, "targetName"] <- sub[1,"target"]
    df[i, "completed"] <- as.numeric(any(sub$choice == sub$target))

    sub <- removeBacks(sub)
    df[i, "traveledPath"] <- nrow(sub)
    sub <- removeCycles(sub)
    df[i, "cyclefree"] <- nrow(sub)
  }
  return(df)
}
```

This reads in all of the three database tables for each of the three networks. It creates two functions to remove cycles (to get the true traveled path) and adjust for using the back button. It then pulls relevant information from each instance of gameplay. It then does various analysis methods and visualizations, such as side-by-side bar plots to view the different completion rates of each network (see Figure 17). It also shows the mean degree for the source and target vertices for each network, separated by if the game (with each source and target pair) were completed (see Figure 18). Many of the tables were generated using similar code. Here is an example that generates the average shortest path between the source and target nodes, separated by network and by completion (see Table 2):

```
avgPath <- function(results){
  comp <- results[results$completed == 1, ]
  uncomp <- results[results$completed == 0, ]
  return(c(mean(comp$realPath), mean(uncomp$realPath)))
}
t <- matrix(NA, ncol = 2, nrow = 3)
colnames(t) <- c("Completed", "Not Completed")
rownames(t) <- c("Base Network", "Random Network", "Smart Network")

t[1,] <- avgPath(baseResults)
t[2,] <- avgPath(randomResults)
t[3,] <- avgPath(smartResults)
t
```

Acknowledgements

While this thesis was written in a semester, the processes that contributed to my completion of a major in Applied Math stretches far before September.

My friends at Yale are a significant part of the reason I stuck with the major. Thanks to Michael Menz, Will Cai, Dave Hatch, & Evan Green for constantly reassuring me that I have what it takes to be an Applied Math major, even when I doubted it and stayed up for hours making no progress on a problem set. Additional thanks to Dave Hatch for assistance in setting up the MySQL database for this project. Thanks to Mari Kawakatsu for loving networks just as much as I do. Thanks to Sophia Kecskes and Julia Carnes for always volunteering to play my Shiny games, even if they were 'really, really, REALLY bad' at them.

Thanks to Dan Spielman for making the Applied Math major at Yale a welcoming and inclusive environment. Thanks to Amin Karbasi helping me decide on a thesis idea and giving me insightful suggestions along the way. Thanks to Michael Frame for serving as a supportive mentor since I began my freshman year.

Thanks to my high school math teachers, especially Max Schoenberg & Daniel Hall, for helping to cultivate a passion for math and programming in my early academic career.

Thanks to Paige Kaliski & Ushasi Naha for not only being great friends and Math Team teammates, but for making me not feel so self-conscious about being a girl who likes math.

Thanks to my parents Mark Tebbe & Patti Pears for giving me logic puzzles & brain teasers from a young age and, in the process, passing down their love for computer science. And thanks for Steve Tebbe for doing those puzzles with me, even geeking out about board games to this day.

Special thanks to Evan Green for always celebrating my successes and sharing my struggles with me, both inside and outside the classroom. I am forever grateful for his help on countless math problems sets (even if he was not in the class), suggesting ways to make my code more efficient, sharing my passion for Plotly, making me laugh when I was (likely unnecessarily) worrying, and for being an overall guru.

References

Amazon Web Services (AWS) - Cloud Computing Services. (n.d.). Retrieved December 1, 2016, from https://aws.amazon.com/

Base Network (n.d.). Retrieved December 1, 2016, from http://ktebbe.shinyapps.io/network1

CourseRec (n.d.). Retrieved December 1, 2016, from http://ktebbe.shinyapps.io/CourseRec

Hamers, L., Hemeryck, Y., Herweyers, G., Janssen, M., Keters, H., Rousseau, R., & Vanhoutte, A. (1989). Similarity measures in scientometric research: The Jaccard index versus Salton's cosine formula. *Information Processing & Management*, *25*(3), 315–318. https://doi.org/10.1016/0306-4573(89)90048-4

Hill, K. (2012, February 16). How Target Figured Out A Teen Girl Was Pregnant Before Her Father Did. Retrieved December 1, 2016, from http://www.forbes.com/sites/kashmirhill/2012/02/16/how-target-figured-out-a-teen-girl-was-pregnant-before-her-father-did/#a7f2d7e34c62

Pazzani, M. J., & Billsus, D. (2007). Content-Based Recommendation Systems. In *The Adaptive Web* (pp. 325–341). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-72079-9_10

Random Network (n.d.). Retrieved December 1, 2016, from http://ktebbe.shinyapps.io/network2

Related Keywords (n.d.). Retrieved December 1, 2016, from http://ktebbe.shinyapps.io/words

Schafer, J. B., Frankowski, D., Herlocker, J., & Sen, S. (2007). Collaborative Filtering Recommender Systems. In *The Adaptive Web* (pp. 291–324). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-72079-9_9

Shiny. (n.d.). Retrieved December 1, 2016, from https://shiny.rstudio.com/

Smart Network (n.d.). Retrieved December 1, 2016, from http://ktebbe.shinyapps.io/network3

Stopwords. (n.d.). Retrieved December 1, 2016, from http://www.ranks.nl/stopwords

West, R., & Leskovec, J. (2012). Human wayfinding in information networks. In *Proceedings of the 21st international conference on World Wide Web* (pp. 619–628). ACM. Retrieved from http://dl.acm.org/citation.cfm?id=2187920

Yale University Developer Portal. (n.d.). Retrieved December 1, 2016, from https://developers.yale.edu/documentation