Dartmouth Computer Science Technical Report TR2015-770

Course InSight: An Application of Natural Language Processing in a Course Browser

Chuanqi Sun
Advised by Professor Xia Zhou

1. Abstract

Course inSight integrates a course catalog with natural language processing and cloud database to provide recommendations based on the concepts extracted from a given course, revealing meaningful connections between courses that are not available through existing technologies. This paper justifies the rationale behind the project, narrates the design process, and discusses implementation details and possible improvements.

2. Introduction

Course election is an integral part of students' educational experience. It involves decision makings that can lead students to new realms of knowledge and success in future careers. While school registrars compile comprehensive catalogs each year to aid the process, students, however, are often overwhelmed by the enormous amount of course offerings, vexed by the meaning of course titles, biased by their peers' word of mouth, and suffered from outdated and impersonal tools. When well-informed course election becomes impractical, students might simply elect courses to raise their GPA, reduce workload, and fulfil requirements for graduation. Dartmouth College, for example, requires students to take classes in a distributive manner, covering nearly all aspects that the college has to offer. However, this liberal arts ideal is undermined by its execution. It is well known that students have compiled a "lay-up list" [1]—a list of distributive classes where nearly everyone gets an "A" without doing much work. In the meanwhile, fraternities at Dartmouth have allegedly compiled a database of all previous exams for some courses so the members can easily pass exams without "wasting" their time in libraries. Although these problems can not all be contributed to an ineffective course election system, it is justified to trace them to the very beginning of the process, course browsing, where we can possibly take proactive measures to encourage well-informed course election.

3. Existing solutions

The surveyed institutions have strikingly similar course browsers. Stanford [2], MIT [3], UPenn [4], Dartmouth [5], Princeton [6], Brown [7], and Columbia [8] all share the same design—a digital catalog indexed by department names or field of studies, searchable by keywords, instructor name, course title, and course period. Massive Open Online Course platform such as Coursera [9], Udacity [10], MIT OpenCourseWare [11], and Stanford Online [12] offer more features including multimedia and social network widgets. All of the aforementioned course browsers, however, organize courses in the structure of a list, which is what "catalog" by definition does. In most cases, the course browser is simply a digitized version of the printed catalog. To summarize, we point out four dangerous assumptions made by the existing solutions:

1. The browser assumes that each user has already come up with a goal to search for, whether it's a keyword, a professor's name, or a course title. The database is indexed and structured with these search goals in mind but if a user would literally like to "browse" courses, the tool would be of no help.

2. The browser assumes a linear structure of courses within a department. In most cases, the default view is sorted by course numbers, which supposedly reflects the progression in difficulty and the deepening and narrowing of topics. This assumption breaks when a department organizes its courses according to its own rule. Take the Computer Science department at Dartmouth for example, the bulk of the courses are divided into three sections: theories (30-49), systems (50-69), and applications (70-89). In many cases, students would take a higher numbered course before a lower numbered one. Sorting these courses into a list according to their numbers would only mislead students.

3. The browser assumes that each department is independent of others. Although many tools nicely present courses offered by one department, they have to display redundant language for courses that are "cross-listed" under two or more departments, let alone the waste of storage and the difficulty in maintaining consistency. As we see more and more interdisciplinary studies and cross-departmental collaborations, the organization of courses by department will soon be outdated.

4. The browser assumes that students are the only users and that information flows strictly from faculty to registrar and then to students. What if faculty would like to get feedback on their posted courses? What if administration would like to measure faculty's performance? What if students would like to know what their peers think of a course? Admittedly, each school has its own feedback system but instead of asking users to take surveys and wait for the analysis to surface after much delay, the data could be collected as soon as the course browsing or shopping period begins and be analyzed and presented in real time.

A successful solution therefore will have to address the following challenges: 1) providing a guidance for course browsing 2) organizing courses to reflect internal relationships rather than by numbers 3) presenting courses across departments and disciplines 4) collecting, analyzing, and presenting user behavioral data.

4. Design

A course browser is essentially a visualization of course-related data. Hence, we tackle the design problem with a data-driven mindset—what aspects of the data should we present, and how to present them effectively? The only data source we can leverage is Organization, Regulations, and Courses (ORC) website [5], which has a catalog of all the courses. On ORC, each course is archived into a single web page with a URL. The page contains the title of the course, its number, a paragraph of description, the name of the instructors, the Distributive and/or World Culture requirements it satisfies, and the term and time period it is offered. The ORC itself does a decent job presenting these fields of data but in order to guide a user through the massive number of courses and visualize connections between them, we must "understand" what each course is about. Once we "understand" the concept of each course, we can establish relationships between courses sharing similar concepts. We may even allow a user to browse directly from a concept. Therefore we treat the title of the course and its description as our primary source of data, from which concepts can be extracted. The goal has now become the visualization of concepts for each course, and the visualization of concept-based relationship between courses.

To further develop this idea, the browser should present two kinds of entities, courses and concepts. Things become interesting when there are two or more of these entities on stage: a course may cover a few concepts; many courses may share the same concept; two courses may cover a very similar set of concepts; some concept is covered by many courses; some courses cover a broad range of concepts... All these seemingly complicated relationships can be reduced to one simple syntax: "course S covers concept T". If we model a database after this syntax, we can answer all the questions of interest by a few database operations. For example, to find the courses that are related to S1, we have the following algorithm:

1. Find all the concepts that S1 covers.
2. For each of the concepts, find all the covering courses.
3. For each of the covering courses, if it has not appeared before, add it to the result and store the relevance value, otherwise, add the relevance value to the previous result.
4. Sort all the results in descending order of their aggregated relevance value.

Another criterion for relating one course to another is by prerequisites. The ORC displays the prerequisites for any given course in the format of "course S1 uses S2 as a prerequisite", but does not show what courses use the given course as a prerequisite. Since prerequisites are "many

to many" relationships, the query from either direction would work. For example, "select all prerequisites where S1==Sx" yields all the courses that use Sx as a prerequisite. On the other hand, "select all prerequisites where S2==Sx" yields all the courses that are used by Sx as a prerequisite.

With the underlying data model, we specify the following key features for the browser:

1. The browser shall handle queries on courses using traditional parameters i.e. title, description, and course number.
2. The browser shall present related concepts for any chosen course.
3. The browser shall present courses covering any chosen concept.
4. The browser shall present relating courses based on prerequisites or shared concepts.

The most intuitive visualization of these features would be a graph where each course or concept is represented as a node and each relationship between a course and a concept or between two courses is represented as an edge. This design is effective only when the number of nodes is small. As the number of nodes increase, the network will become so dense that it's neither computationally affordable nor visually appealing (Figure 1).

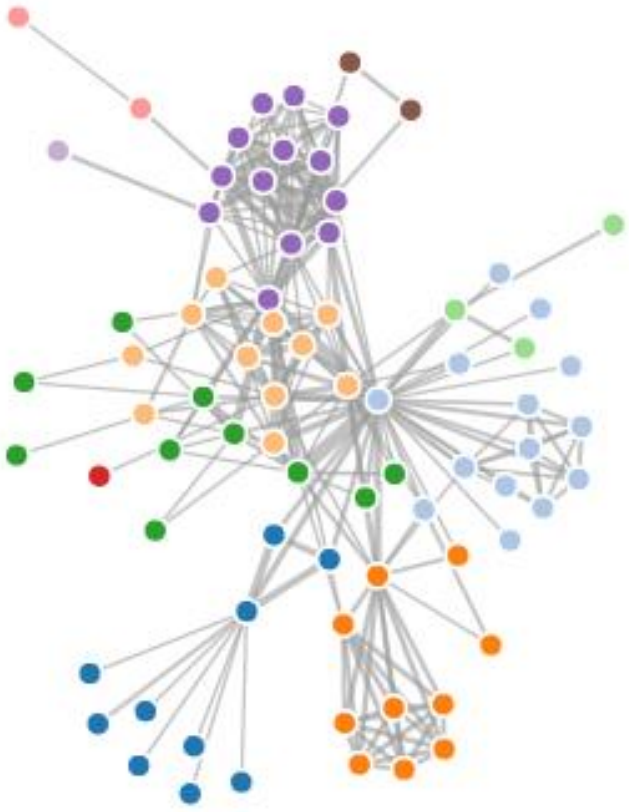

Figure 1 – Network Graph[1]

[1] Image credit: Image Credit: http://tallytree.com/wp-content/uploads/2012/07/D3_force_directed_layout.png

To address this issue, we propose a dynamic hierarchical data structure where the root node always represents the course or concept of interest. When the user navigates from a course to a concept, or vice versa, or from one course to another, the root node is updated to reflect the change. With this technique, we are able to limit the depth of the hierarchy to a constant number, in which case a sunburst graph (Figure 2) would be the ideal visualization.

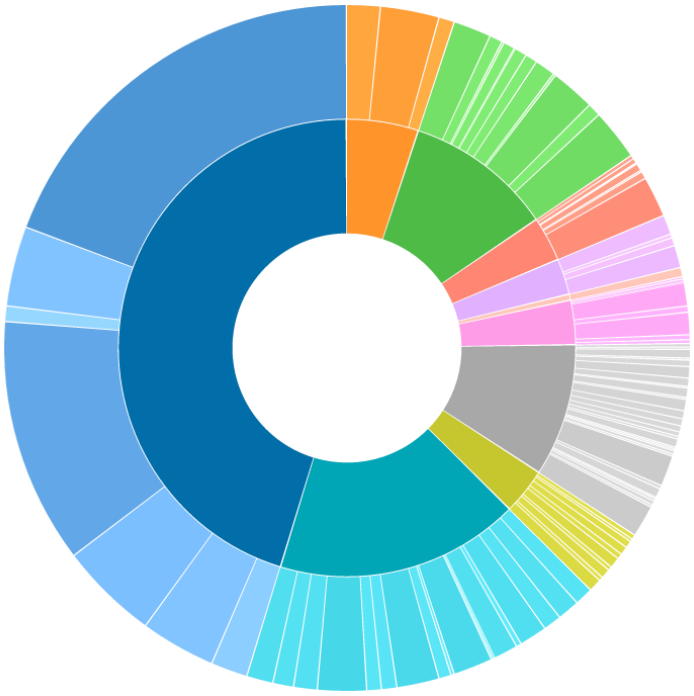

Figure 2 – Sunburst Graph[2]

5. Implementation

5.1. Data Collection

The ORC organizes the URL's of the courses into a tree structure with a depth of 3—the home page being the root, each department being a child of the root, and each course being a child of a department. However, there are exceptions for departments with sub-departments (e.g. AMEL), and for departments further dividing courses into sections (e.g. ENGL). Let us denote the following string with [rooturl] "http://dartmouth.smartcatalogiq.com/en/2014/orc/Departments-Programs-Undergraduate/", then there are three possible formats for the URL of a course:

1. [rooturl]/[course number]
2. [rooturl]/[sub department name]/[course number]
3. [rooturl]/[section name]/[course number]

[2] Image credit: http://stackoverflow.com/questions/24438313/stop-zooming-or-folding-in-sunburst

We can simplify the URL's into one format: [baseurl]/[course code] where [baseurl] has the three variations above. Given that there are only 84 unique [baseurl] strings in ORC, it is faster to collect them by hand than by writing, debugging, and running a script. Once we obtain the [baseurl], we can use a crawler to visit each base URL and extract course URL's from the web page. With URL's for all the courses, we then use another crawler to download the pages and extract the course number, title, prerequisite, time period, and professor from the HTML file and store them in JSON format. Now the data is ready to be loaded into a database. The complete data collection script is listed in the appendix.

5.2. Database

We use Parse [13] to host the database. Parse is an integrated platform for web hosting, database, and cloud computing. We will discuss its cloud functions in later sections. For our application, we maintain 4 tables on Parse:

1. Course: each row stores information about one course: title, number, URL, time period, professor, distributive requirements, and description.

2. Prerequisite: each row specifies a dependency—one course is used by another as a prerequisite. We use the pointer data type provided by Parse to link the two courses in each prerequisite relationship to their corresponding rows in the Course table.

3. Concept: it stores the concepts relevant to each course, along with a relevance value. The higher the value, the more relevant the concept is.

4. User Activity: it stores the user behavioral data. It will be discussed in section 5.4 and 5.6.

Parse provides a comprehensive JavaScript API for adding and updating content in a database. To populate the Course, and Prerequisite tables, we use a node.js application to iterate the courses and prerequisites in the previously exported JSON file and uploads them to the database with the JavaScript SDK. The other two tables will be populated in later steps. The complete upload script is listed in the appendix.

5.3. Concept Extraction

Concept extraction is a well-researched topic in natural languages processing (NLP). We decide to use AlchemyAPI [14], an online NLP SaaS that parses raw text into concepts. Note that the NLP algorithm is not perfect so not all concepts are truly relevant to a course. We will have to use "relevance" value to filter away irrelevant results. When

finding a relevant course, we require that two courses share one or more concepts the relevance values of which add up to a certain threshold. There are 2228 courses in ORC. If we were to find all courses that share concepts with a given course, we would have to issue 2227 API calls, which would hit the API cap, not to mention the time it would take to process them. The solution is preprocessing all the courses to have their concepts extracted and stored in our database on Parse. Again, we use a node.js application to iterate the courses in the previously downloaded JSON file, extract the concepts with an npm package for AlchemyAPI, and uploads the results onto Parse using its SDK.

5.4. Cloud Functions

In a typical JavaScript web application, the client, usually a web browser, is responsible for computing the data once they are fetched from a database. There will be a significant penalty on performance when the desired results are only a few rows but the computation requires the knowledge of the entire table, or worse still, two or more tables. For example, when we compute the most viewed courses, we have to aggregate user activity counts on the keyword "course" and rank the results in descending order of the aggregated counts. Although we could let the database alone handle this complicated query, it would be very difficult to run customized filters on top of it. But if we download the entire user activity table to client side and does the computation, the download would take too long. The cloud function combines the advantages of the two by running JavaScript on the database server. To get the most viewed activities, we simply call the cloud function through Parse API, and the result will be returned instantly.

Another use of the cloud function is event handling. Similar to what a "trigger" does in traditional database, a cloud function can be triggered when a row is added or updated. In our application, a cloud function is executed after each CREATE or UPDATE event in the Prerequisite table. The function sets a pointer from a course in Prerequisite table to its counterpart in the Course table. We also use a cloud function to create a lowercased copy of a concept when it is added to the database to facilitate search.

5.5. User Interface

We built the front end of this application with D3.js [15] framework. D3.js provides a set of visualization templates that can be directly applied to arrays of objects in JavaScript, which in our case can be easily fetched from Parse using its SDK. As we previously discussed, we use sunburst graph to visualize the dynamic hierarchical structure. Two different views will be covered:

1. Course View (Figure 3)

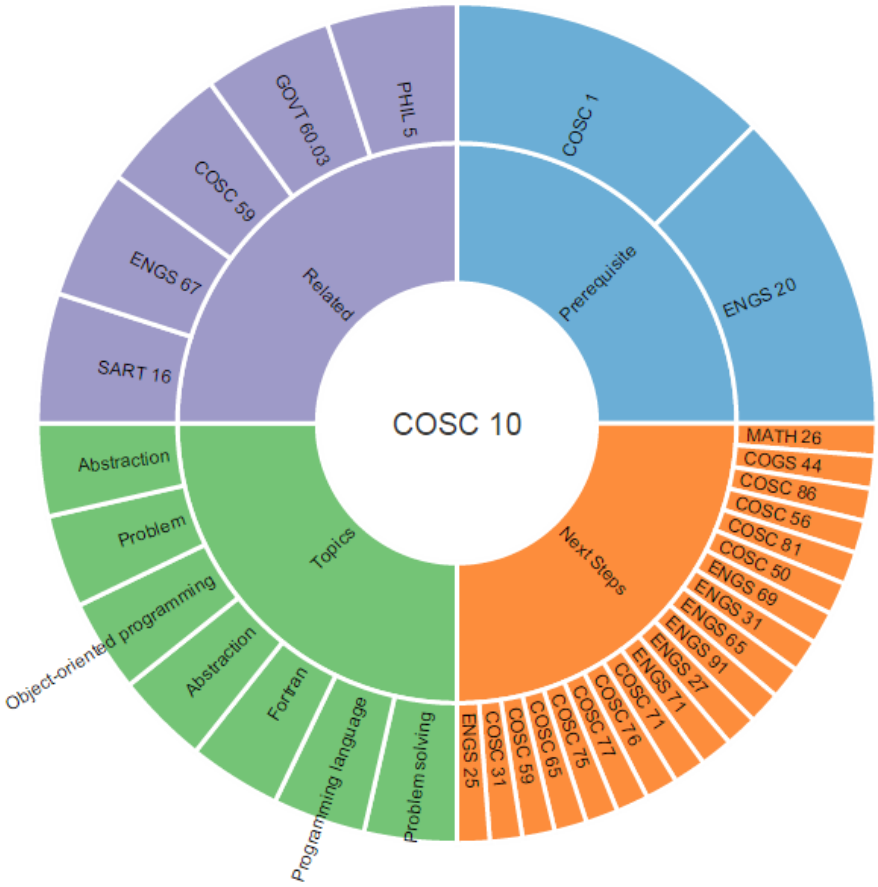

Figure 3 – Course View

In Course View, the center of the sunburst graph is set to the course of interest. Surrounding the center are four sections: 1) "Related": courses that are related to the course shown in the center. The relationship is determined by shared concepts. 2) "Topics": concepts extracted from the course shown in the center. 3) "Prerequisite": courses that should be taken prior to the course shown in the center. 4) "Next Steps": courses that use the course shown in the center as a prerequisite. The sunburst graph displays the information of interest in a minimalistic view—only course numbers or concept names are displayed. To compensate the lack of details, we set up a side panel to display the full length information when user mouses over certain areas on the sunburst graph (Figure 4). When user clicks on a course or concept, the graph will be updated to the corresponding view with the new course or concept in the center.

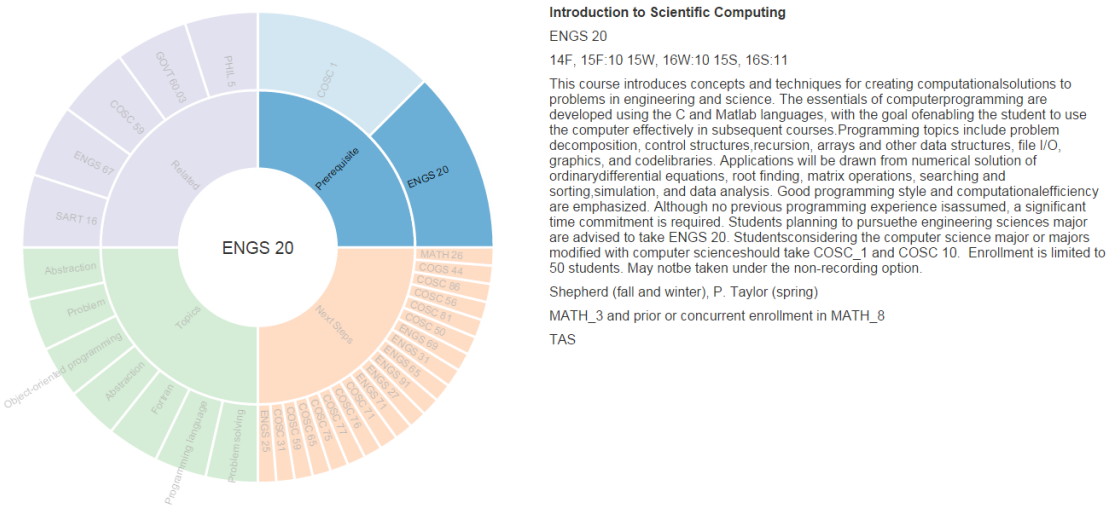

Figure 4 – Course View with side panel

2. Concept View (Figure)

The Concept View is similar to the Course View except that the courses related to a concept is displayed directly around the central circle. When there are many courses covering a concept, the surrounding sections will become very narrow (Figure 5).

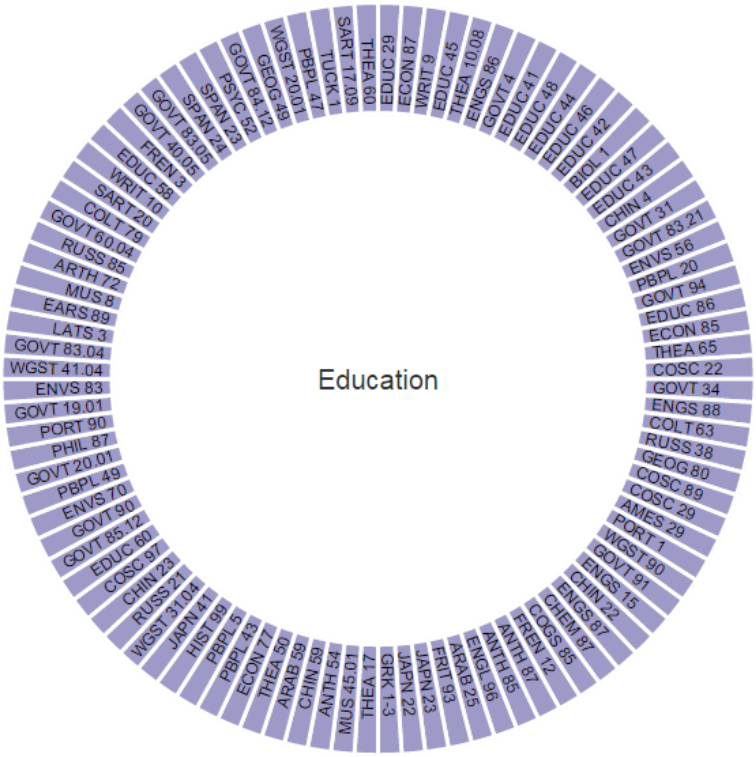

Figure 5 – Concept View

Again, a side panel is used to display full length information when user mouses over a course but in order to minimize eye movement, we display the enlarged course number in the center of the sunburst, a design trick that has already been adopted in the Nest Learning Thermostat [16].

The application features a search box on the top of the screen in which users can search either a concept or a course. Powered by typeahead.js [17], the search box updates results as the user types. We treat the keywords as part of a course title, a course number, and a concept all at the same time and display all of the results on the same page so the interface will not confuse the user with complicated search options (Figure 6).

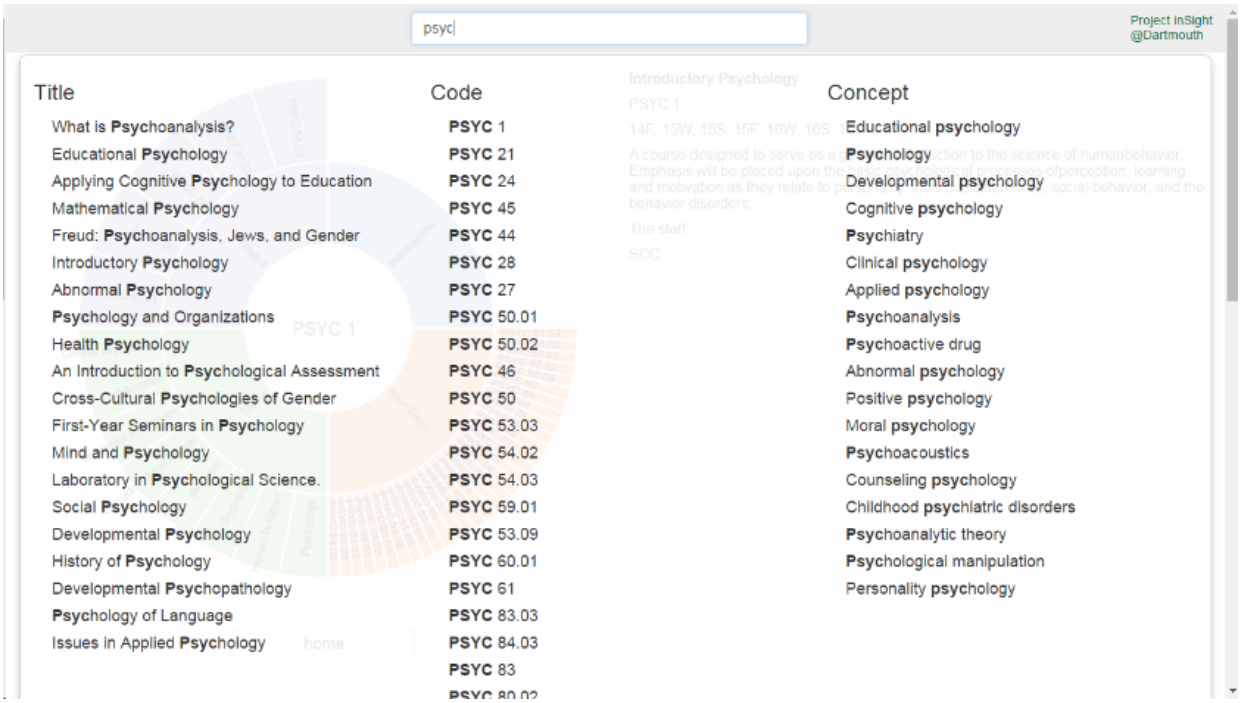

Figure 6 - Search Results

The homepage of the application is composed of the search box previously described and two word clouds taking up the rest of the screen (Figure 7). When the user mouses over a course or a concept, the word is enlarged as a visual feedback of selection. Once clicked, the word will take the user to the corresponding Course View or Concept View. The word clouds are generated with a D3.js plug-in using the most popular courses and concepts. The popularity is determined by recorded user activity, which we will discuss the next.

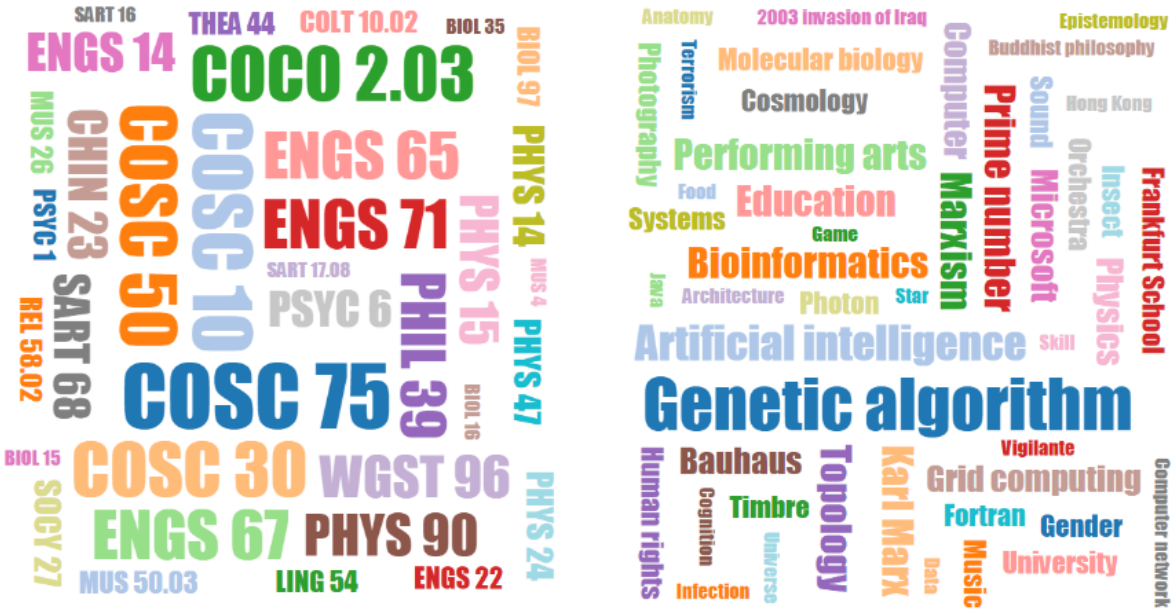

Figure 7 - Homepage

5.6. Telemetries

Two kinds of behavioral data are collected. Search and exploration. When a user transitions into a course or a concept from the search box, a search activity is logged along with the keyword that is used in the search and the time of event. If the transition originates from a word cloud or from an existing sunburst graph, an exploration activity is logged along with the course name or the concept name and the time of event. On one hand, this unobtrusive technique of data collection has completely freed the users from filling out forms and revealing private information. On the other hand, the data collected is limited to crowd behaviors. If individual behavior needs to be collected, we will have to establish user profiles.

6. Conclusion and Future Work

This project is a comprehensive exercise of data mining strategies, database design, cloud computing, natural language processing and UI/UX designing. The outcome is a user friendly web portal hosted at dartmouth.parseapp.com. The portal helps students find courses based on their interests and achieve insights in how one course will lead to another. Thanks to cloud hosting and cloud computing technologies, this project can be deployed without the requirement of a dedicated web server or database host. The user interface is loosely coupled with the backend by API calls so it adapts well to other course databases. We identify the following aspects of the project that can be improved by future work.

6.1. User Profiling

In the current version of the application, each user remains anonymous throughout the whole session. Establishing a profile for each user would benefit not only the users by enabling browsing history, bookmarks on favorite courses, and sharing findings on social network, but also the faculty and administration by providing individual and yet anonymous behavior for further analysis and revealing patterns and trend during course election. Above all, telemetry with profiled users is more accurate and representative. To establish user profiling, the database needs to be upgraded to include a user table. The application needs to include a login and logout function, as well as session management. An ideal solution would integrate into a Single Sign-on system (SSO) that a school usually provides. The benefit of using SSO is 1) hassle-free identity management because profiles can be pulled directly from school directory 2) better access control because the school has full control over what content can be viewed by student, faculty, and administration.

6.2. Database Auto-maintenance

As described in 5.1 and 5.2, the Course and Concept tables are populated by one-time data crawled from ORC. Whenever ORC updates, the crawler has to run again, after which the tables have to be dropped and re-populated. The user activities may also introduce inconsistency if an old course in the activity no longer exists in the new ORC website. The proposed solution is running a crawler that periodically check for updates on ORC. If a course changes, a series of actions are triggered to maintain the integrity of the data. The auto-maintenance can be implemented with a combination of crawler scripts for downloading course pages, node.js scripts for finding changes and uploading changes, and Parse cloud functions for maintaining integrity.

6.3. Search Filters

Existing course browsers usually support various filters to help users narrow down the results. ORC website, for example, allows users to filter by term, time period, department, distributive requirement, and world culture requirement. The major challenge in implementing these filters is building a robust HTML parser that can handle various mistakes on ORC when converting HTML into numerical or enum types. Even with a perfect parser, the language on ORC is too haphazard to be machine readable—there are prerequisites that either can be taken concurrently, or require instructor's permission, or can be fulfilled by a course from a different department. There are also courses that either are offered by arrangement, or have multiple sections offered at the same or different periods, or can be taken multiple times. After all, we doubt the rules and regulations can

be perfectly modeled by logic without human intervention, at least not with the available technology. The best solution would be obtaining native access to ORC database.

7. Acknowledgement

I would like to extend my deepest gratitude to my thesis advisor Professor Xia Zhou for offering invaluable suggestions and guidance in the course of this project. I also thank Professor Devin Balkcom and Computer Science Department administrator Joseph Elsener for their timely help in logistics. In the meanwhile, I thank all the people who tested the application during the early stage and gave me constructive feedback through social media or in person. For any questions, please contact the author at henrysun918@gmail.com.

8. References

[1] J. Asch, "The Layup List," 8 2 2013. [Online]. Available: http://www.dartblog.com/data/2013/02/010644.php.

[2] "Stanford Bulletin EXPLORE COURSES," [Online]. Available: https://explorecourses.stanford.edu/. [Accessed 3 5 2015].

[3] "MIT Course Picker," [Online]. Available: https://picker.mit.edu/. [Accessed 3 5 2015].

[4] P. Labs, "PENN COURSE REVIEW," [Online]. Available: https://penncoursereview.com/. [Accessed 3 5 2015].

[5] "Organization, Regulations, and Courses 2014," [Online]. Available: http://dartmouth.smartcatalogiq.com/en/2014/orc/Departments-Programs-Undergraduate. [Accessed 3 5 2015].

[6] "Office of the Registrar Course Offerings," [Online]. Available: https://registrar.princeton.edu/course-offerings/. [Accessed 3 5 2015].

[7] "Brown University Courses," [Online]. Available: https://courses.brown.edu/. [Accessed 3 5 2015].

[8] "Columbia University Directory of Classes," [Online]. Available: http://www.columbia.edu/cu/bulletin/uwb/. [Accessed 3 5 2015].

[9] "Coursera," Coursera Inc., [Online]. Available: https://www.coursera.org/courses. [Accessed 3 5 2015].

[10] "Nanodegrees and Courses," Udacity, Inc., [Online]. Available: https://www.udacity.com/courses/all. [Accessed 3 5 2015].

[11] "Courses," Massachusetts Institute of Technology, [Online]. Available: http://ocw.mit.edu/courses/. [Accessed 3 5 2015].

[12] "Standord ONLINE UPCOMING & IN SESSION," Stanford University, [Online]. Available: http://online.stanford.edu/courses. [Accessed 3 5 2015].

[13] "Parse," Parse, [Online]. Available: https://parse.com/. [Accessed 3 5 2015].

[14] "Powering the New AI Economy," AlchemyAPI, Inc., [Online]. Available: http://www.alchemyapi.com/. [Accessed 3 5 2015].

[15] M. Bostock, "D3 Data-Driven Documents," [Online]. Available: http://d3js.org/. [Accessed 3 5 2015].

[16] M. Rogers, "What's in the 4.3 software update," Nest Labs, 4 11 2014. [Online]. Available: https://nest.com/blog/2014/11/04/whats-in-the-4-3-software-update/.

[17] "typeahead.js," Twitter, Inc., [Online]. Available: https://twitter.github.io/typeahead.js/. [Accessed 3 5 2015].

9. Appendix

The whole project is hosted on GitHub at https://github.com/henrysun918/cloud-nlp. Instead of listing source code, the appendix will refer to hosted files.

Hand compiled [baseurl] file: https://github.com/henrysun918/cloud-nlp/blob/master/offline/data/base_urls

Script to crawl course URL's from [baseurl]: https://github.com/henrysun918/cloud-nlp/blob/master/offline/data/mine.sh. The output of the script: https://github.com/henrysun918/cloud-nlp/blob/master/offline/data/urls.txt

Script to crawl course data given course URL's: https://github.com/henrysun918/cloud-nlp/blob/master/offline/data/load_json.sh. The output of the script: https://raw.githubusercontent.com/henrysun918/cloud-nlp/master/offline/data/course.json

Script to extract concept through AlchemyAPI and upload course data in JSON file to Parse database: https://github.com/henrysun918/cloud-nlp/blob/master/offline/app.js.