

7-12-2014

Visualization of Student Cohort Data With Sankey Diagrams via Web-Centric Technologies

Caleb Morse

Follow this and additional works at: https://digitalrepository.unm.edu/ece_etds

Recommended Citation

Morse, Caleb. "Visualization of Student Cohort Data With Sankey Diagrams via Web-Centric Technologies." (2014).
https://digitalrepository.unm.edu/ece_etds/180

This Thesis is brought to you for free and open access by the Engineering ETDs at UNM Digital Repository. It has been accepted for inclusion in Electrical and Computer Engineering ETDs by an authorized administrator of UNM Digital Repository. For more information, please contact disc@unm.edu.

Caleb D. Morse

Candidate

Engineering

Department

This thesis is approved, and it is acceptable in quality and form for publication:

Approved by the Thesis Committee:

Gregory Heileman

, chairperson

Chaouki Abdallah

Christopher Lamb

Terence Turner

**VISUALIZATION OF STUDENT COHORT DATA WITH
SANKEY DIAGRAMS VIA WEB-CENTRIC
TECHNOLOGIES**

by

Caleb D. Morse

B.S., Computer Engineering, University of New Mexico, 2011

M.S., Computer Engineering, University of New Mexico, 2014

THESIS

Submitted in Partial Fulfillment of the
Requirements for the Degree of

**Master of Science
Computer Engineering**

The University of New Mexico
Albuquerque, New Mexico

May 2014

VISUALIZATION OF STUDENT COHORT DATA WITH SANKEY DIAGRAMS VIA WEB-CENTRIC TECHNOLOGIES

by

Caleb D. Morse

B.S., Computer Engineering, University of New Mexico, 2011

M.S., Computer Engineering, University of New Mexico, 2014

Abstract

Universities have long been analyzing student cohorts in an attempt to understand the factors influencing whether a student continues in their major, switches to another major (often referred to as swirl), or drops out entirely. A visual representation of how students flow between majors and colleges can aid in this understanding. This thesis will study how a student cohort can be represented in a Sankey Diagram. In this analysis, a student cohort is represented as a weighted, directed, graph, where a vertex represents a distinct class status for a semester and each weighted edge represents the number of students moving between class statuses. To ease understanding, this graph is represented visually as a Sankey Diagram, a special kind of flow diagram. The purpose of this tool is to give university administration a visual representation of the flow of a student cohort between majors and colleges on a semester-by-semester basis. This representation can be viewed at three levels of granularity: that of an entire university, college, or a single major.

Contents

List of Figures	v
1. Introduction	1
1.1. Background of Problem	1
1.2. Sankey Diagrams.....	1
2. Specific Requirements for Student Flows	3
3. Architecture and Design Choices	4
4. Implementation and Experimentation	7
4.1. Client Implementation.....	8
4.2. Server Implementation	12
4.3. Server Database Schema	17
4.4. Server Handling Client Queries for Vertex Intersection.....	17
4.5. Server Caching Strategies	19
5. Conclusions and Future Work.....	21
Appendices.....	23
Appendix A Supplementary Sankey Diagrams	24
References.....	34

List of Figures

Figure 1 – First flow (Sankey) diagrams made by Riall representing an ideal steam engine vs actual steam engine [2].....	2
Figure 2 – Block Diagram for High-Level Application Design	7
Figure 3 – Client configuration interface.....	8
Figure 4 – Sankey Diagram for Electrical Engineering students in the 2007 cohort.	10
Figure 5 – Sankey Diagram highlighting students who passed through the EN_EE5 for Electrical Engineering students in the 2007 cohort.....	11
Figure 6 – Client Query Validation block diagram	14
Figure 7 – Flow chart for requesting additional information from database	15
Figure 8 – Flow chart for detecting which flow a student is currently part of for the College Group case.....	16
Figure 9 – Database Schema.....	17
Figure 10 – Query response time comparison	19
Figure 11 – Sankey Diagram for all students in the 2007 cohort.	24
Figure 12 – Sankey Diagram highlighting students who passed through the AS5 vertex for all students in the 2007 cohort.	25
Figure 13 – Sankey Diagram for engineering students in the 2007 cohort.	26
Figure 14 – Sankey Diagram highlighting students who passed through the GENG5 vertex for engineering students in the 2007 cohort.	27
Figure 15 – Sankey Diagram for English students in the 2007 cohort.	28
Figure 16 – Sankey Diagram highlighting students who passed through the AS_NON_ENGL5 vertex for English students in the 2007 cohort.	29
Figure 17 – Sankey Diagram highlighting students who passed through the AS_ENGL5 vertex for English students in the 2007 cohort.	30
Figure 18 – Sankey Diagram for Electrical Engineering students in the 2007 cohort.	31
Figure 19 – Sankey Diagram highlighting students who passed through the EN_NON_EE5 vertex for Electrical Engineering students in the 2007 cohort.....	32
Figure 20 – Sankey Diagram highlighting students who passed through the EN_EE5 vertex for Electrical Engineering students in the 2007 cohort.	33

List of Tables

Table 1 – Example output of vertex synopsis on client-side application 12

1. Introduction

1.1. Background of Problem

Humans are poorly suited for working with large heterogeneous datasets, but visual representations of this data can greatly increase understanding by providing succinct overviews. However, these visualizations must be created with care, taking into account the inherent limits of human cognition and perception [1]. Thus, visual analytics are often based on graphic summarizations of a complete dataset.

At public flagship universities, a large number of students comprise the entering freshmen cohort each year, and this cohort creates a myriad of data during their time at the university. This data is a valuable source of information regarding student success. Unfortunately, business intelligence tools are often poorly suited for an academic environment because the types of information used by university administration for decision-making differs greatly from the information used in a traditional business environment.

1.2. Sankey Diagrams

A Sankey diagram is special a kind of flow diagram showing the proportional flow through a system, for example, the distribution of heat, energy, etc. through a physical system. The Sankey diagram is an important aid to eliciting understanding of inefficiencies within a system with numerous inputs and outputs. The diagrams are also been used to improve the efficiency of systems by highlighting areas of focus and allowing for the comparison of any changes made to evaluate their efficacy as shown in Figure 1.

In 1898, Matthew Henry Phineas Riall Sankey created a new kind of flow diagram for a publication to show the thermal efficiency of a real steam engine versus that of an ideal steam engine (Figure 1) [2]. This flow diagram would come to be eponymously named a Sankey Diagram. In the diagram, the width of a stream shows the amount of heat leaving and entering each part of the plant per unit time. The thermal efficiency of a steam engine can be difficult to

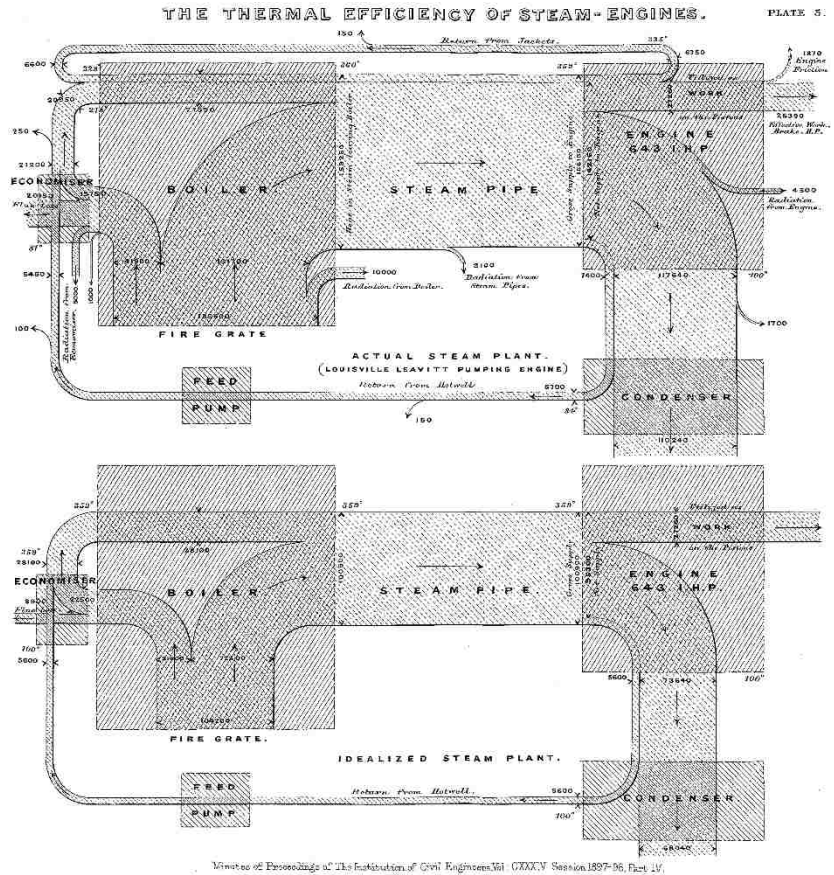


Figure 1 – First flow (Sankey) diagrams made by Riall representing an ideal steam engine vs actual steam engine [2].

understand due to the numerous areas that energy is exchanged. A Sankey Diagram puts this information in a visual format making it easier to understand, even upon a cursory examination.

Historically, Sankey Diagrams had to be drawn manually [3]—a painstakingly slow and potentially error-prone process due to the many approximations and assumptions made in the creation of the diagram. Fortunately, the advent of computers allows Sankey Diagrams to be generated automatically; this also paves the way for substantially larger datasets to be reasonably utilized.

2. Specific Requirements for Student Flows

Thousands of students move through a large university every semester. This movement can include anything from changing majors, changing departments, dropping out, etc. resulting in tens of thousands of data points per student cohort. Thus, it is desirable to visualize these so-called student flows in a manner, which allows questions such as, “is a student more or less likely to change majors after being admitted into the college for that major” to be visually answered by university administration. These types of questions can be answered using existing methods, but representing the data as a Sankey Diagram allows for more organic user-driven exploration without having to request that a new report be written.

A student cohort can be represented as a weighted, directed, graph, where a vertex represents a distinct class status for a semester and each weighted edge represents the number of students moving between two class statuses. This directed graph can be represented visually as a Sankey Diagram representing the major/college that students in a cohort were enrolled in on a semester-by-semester level. The Sankey Diagram will also support three separate levels of granularity: (1) a university-level view (i.e., all students in the cohort and the colleges they are enrolled in); (2) a college-level view (i.e., all students enrolled in a college and their declared major); and (3) a major-level view (i.e., all students with a specific declared major).

3. Architecture and Design Choices

The application was designed in a traditional client-server model. In the context of generating Sankey Diagrams from student cohort data, this model is advantageous for a number of reasons. (1) The quantity of student data necessary to generate the Sankey Diagrams can be quite substantial in size; the size of the 2006 cohort at the University of New Mexico was approximately 3,000 students. The data for 8 semesters on 3,000 students is unsuitable for transmission to the client. (2) The student data required to generate the Sankey Diagrams could potentially contain information of a personally identifiable nature, even in an anonymized form, making it of concern under FERPA (Family Educational Rights and Privacy Act). (3) A substantial quantity of pre-processing is necessary to output the information necessary to render a Sankey Diagram, making the application unusable on many portable devices if a majority of the data processing were done on the client side.

Storage for the server-side application is provided by PostgreSQL¹, a traditional relational database management system. PostgreSQL is used because the student data used in generating the Sankey Diagrams is in a highly structured format, which is ideal for storage and querying in an SQL database. Currently, the database server is resident with the web server; this was done for convenience during development, there is no technical reason that the database server cannot be run on a separate server. The primary work of filtering the student data is performed by the PostgreSQL server with queries generated by a Ruby script. That is, the student data returned by an SQL query is only the data specifically related to the current client query; this way, the Ruby

¹ <http://www.postgresql.org/>

script on the server side need only assign the students to their respective edges. Details on the database schema used are in section 4.3.

Performance of the application on the server-side is a concern; for every request, the server must run one or more complex SQL query and then run post-processing on the query result. With many simultaneous users, performance could quickly degrade. To mitigate this, Redis² (REmote DIctionary Server), a key-value store was used to cache the data associated with previous queries. More details regarding the caching strategy is in section 4.5.

On the client side, the D3.js³ (Data-Driven Documents) JavaScript library is the primary component used. D3 is a visualization-focused abstraction of the SVG⁴ (Scalable Vector Graphics) and DOM (Document Object Model) APIs found in all modern web browsers [4]. This library allows the rendering of the Sankey Diagrams to be done on the client side, greatly simplifying the complexity of the both the server and client-side applications. The D3 library also allows for dynamic user interaction with visualizations; this kind of interaction would not be possible with a scheme using static imagery generated by the server-side application.

Communication between the client and the server is done via HTTP (Hypertext Transfer Protocol), support for which is part of the basic functionality of every web browser, which is why it is used as the application protocol. Support for HTTP on the server side is provided by the Sinatra⁵ project. The data returned to the client is encoded in the JSON (JavaScript Object Notation) format, which was picked because of widespread support for the format in web browsers that also support SVG.

² <http://redis.io>

³ <http://d3js.org>

⁴ <http://www.w3.org/Graphics/SVG>

⁵ <http://www.sinatrarb.com/>

The server component was designed to support deployment on Amazon EC2 (Elastic Compute Cloud) virtual machines. This introduced some variables that would not have otherwise been a factor. The first issue is that the larger an instance is, the greater its per-hour cost. The second issue is that the instance may have significant performance variability due to the noisy neighbor problem. For these reasons, it is important that the server-side application have reasonable response times, even in the face of performance variability. Using Redis-based caching substantially improves request response times, especially for common requests. In practice, the server application was deployed on an *m1.large* EC2 instance. If additional performance is required, EC2 instances such as *m1.xlarge* that are more powerful could be used. Alternatively, the Redis and PostgreSQL servers could be moved to separate instances from the webserver that are better suited for the memory-based and database-style workloads.

4. Implementation and Experimentation

The development of the client component were done in a largely platform-agnostic way, in that any platform which includes a web browser with support for SVG, CSS3 Transitions, and JavaScript is capable of displaying the Sankey Diagrams. All major web browsers have SVG support, including Internet Explorer, Google Chrome, Firefox, Opera, and Safari; including the mobile versions of these web browsers [4]. The client side code has been used under many of these browsers under a variety of operating systems without any issues. Figure 2 shows a high-level block diagram for the client-server application design.

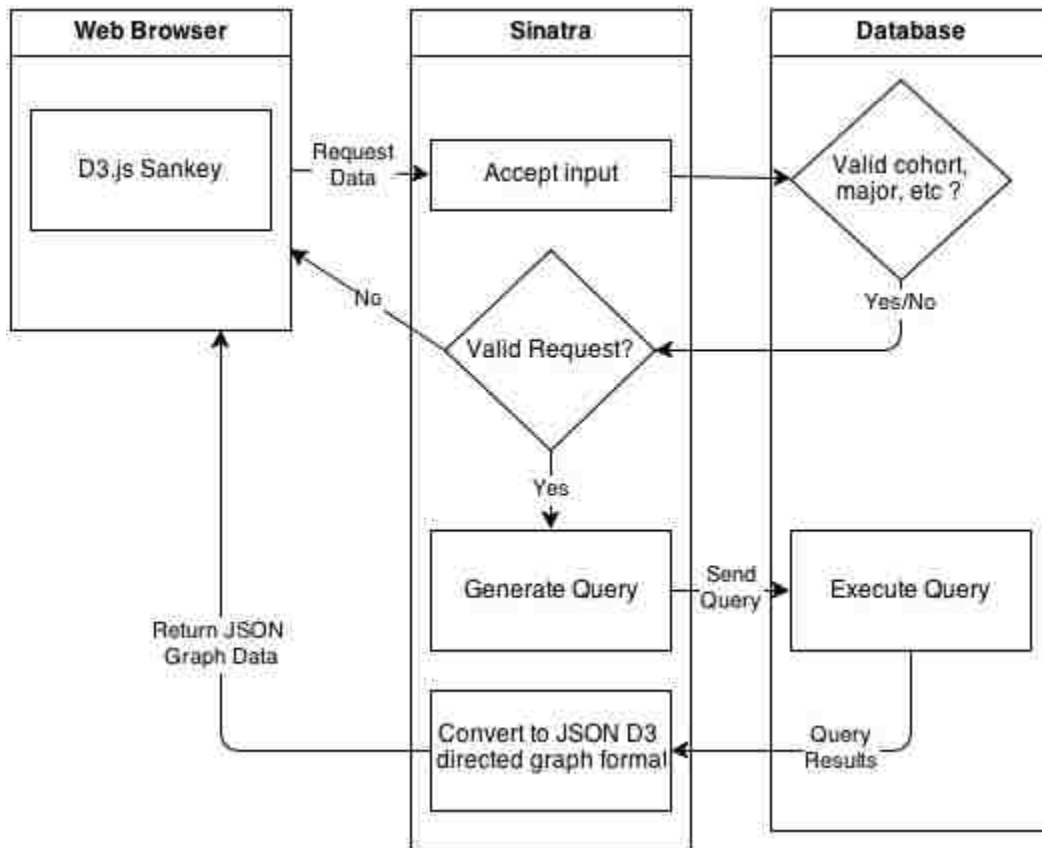
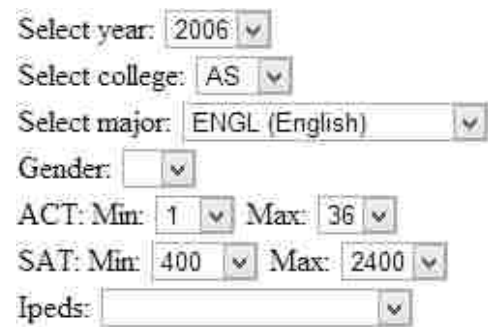


Figure 2 – Block Diagram for High-Level Application Design

The development of the server was done exclusively on Linux (Ubuntu 13.04) with PostgreSQL 9.1 and Ruby 1.9⁶. In theory, the server software component could run on any system that supports a compatible version of PostgreSQL and a minimum of the Ruby 1.9 interpreter, but this has not been tested. The Ruby-based Arel⁷ (A RELational aLgebra) project was used to generate SQL queries. The Arel project includes support for generating queries for MySQL and Oracle databases, so, in theory, it should be possible to use any of the databases supported by Arel in lieu of PostgreSQL.

4.1. Client Implementation

The first time that the client runs the application, it requests a configuration file from the server. This configuration file contains information regarding the cohorts, colleges available for each cohort, and the majors that are available within each department. This configuration information is used to provide an interface



The screenshot shows a web form with the following fields: 'Select year:' with a dropdown menu showing '2006'; 'Select college:' with a dropdown menu showing 'AS'; 'Select major:' with a dropdown menu showing 'ENGL (English)'; 'Gender:' with a dropdown menu; 'ACT: Min:' with a dropdown menu showing '1' and 'Max:' with a dropdown menu showing '36'; 'SAT: Min:' with a dropdown menu showing '400' and 'Max:' with a dropdown menu showing '2400'; and 'I-peds:' with a dropdown menu.

Figure 3 – Client configuration interface

allowing for the selection of any valid combination of cohort year, college, and major. An example of this interface is in Figure 3. A more detailed description of how this configuration file is created can be found in section 4.2.

After loading the configuration file, the user can then select a combination of the cohort year, and optionally a college and major. A major can only be selected if a college has been selected. If a major is not selected (i.e., the user selected the “ALL” option), then the view will display all of the possible majors within the selected college for the given cohort year. Similarly,

⁶ <https://ruby-lang.org/>

⁷ <http://github.com/rails/arel>

if a college is not selected, then the view will display all of the possible colleges for the given cohort year.

After an appropriate selection has been made, the client will send a request to the server for the desired information. This request must include the selected cohort year and college (Details on how the server creates a response to this request can be found in section 4.4). The data that the server returns to the client is encoded as a JSON object that contains a list of the vertices and a list of the edges and their respective weights for the weighted, directed, graph that makes up the Sankey Diagram. Vertex positioning information can be calculated automatically on the client-side using the Gauss-Seidel method of iterative relaxation [5], but this algorithm has a runtime complexity of $O(n^3)$ [6]. Some of the graphs at the coarsest level of granularity (e.g., graphs at the University-level and those at the college-level) contain in excess of 200 vertices and 800 edges. Data from an earlier cohort or a larger university would result in a substantially larger numbers of vertices and edges. In such cases, the algorithm's complexity creates a noticeable delay on the client in displaying the diagram. Each vertex contains information on vertical and horizontal (i.e., which semester) positioning.

Once the client has received data from the server, boxes are created that represent each of the vertices whose vertical height is proportional to the number of students that pass through that vertex relative to the total number of students in the cohort as shown in Figure 4. The labelling of the vertices varies depending on the current view of the Sankey Diagram. For all vertices, an integer is appended to the end of the name, representing the number of semesters since the cohort began, where the fall is counted as the first semester. For all views, “*STOP*” and “*GRAD*” vertices are included, each representing students dropping out and graduating, respectively. For the university-level view, the vertices are named for every college available in the university. For the

college-level view, the vertices are named for every possible major available in the college. For the major-level view, the vertices are named for the major in the current college as well as the major within zero or more secondary colleges (e.g., University College). For the college-level and major-level views, additional “*OUTSIDE*,” “*GRAD_OUTSIDE*,” and “*STOP_OUTSIDE*” vertices are added. These vertices represent students who are outside of the selected college, major, or secondary college, and whether the students is currently enrolled, has graduated, or has dropped out.

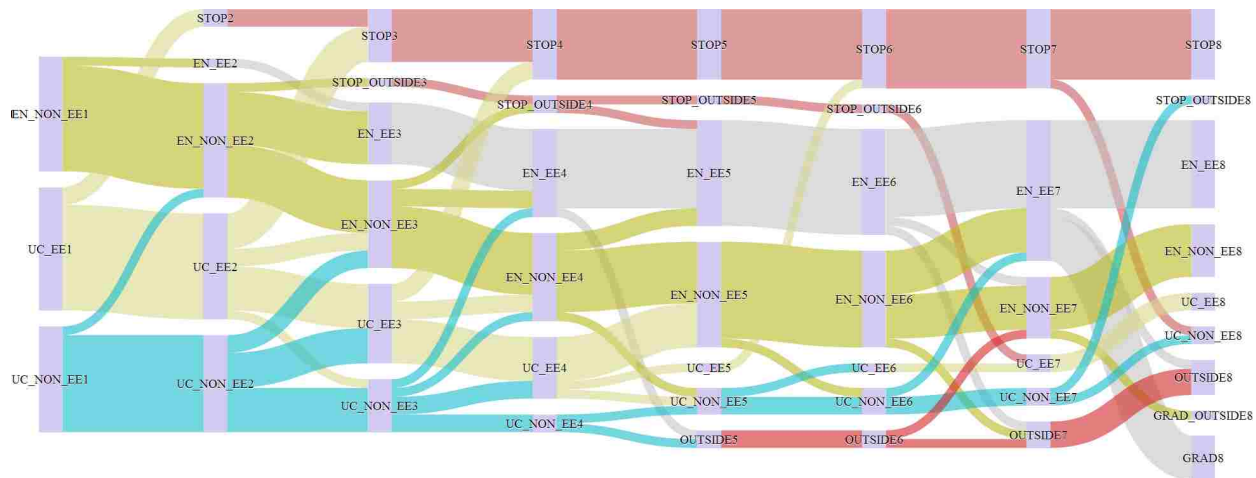


Figure 4 – Sankey Diagram for Electrical Engineering students in the 2007 cohort.

The client application then draws the edges between their respective vertices, with the width of the edge being proportional to the number of students moving through that edge. The colors for the edges are generated from a custom color palette such that the color is the same for every edge with a starting vertex that has the same name, excluding the semester number. Currently 20 unique edge colors are supported in addition to the “*STOP*” and “*GRAD*” edges, which are always red and green, respectively.

Once the initial Sankey Diagram has been rendered by the client (as seen in Figure 4), the client can highlight the paths of students who moved through a particular vertex (e.g., all students

in the 2008 cohort, who majored in Electrical Engineering, and were enrolled in the School of Engineering for their 5th semester). This behavior is triggered by the user hovering their cursor over a vertex. When this occurs, the client sends a vertex intersection request to the server. The client then overlays these new edges on the existing edges, but with a height that is proportional to the weight of the original edges. Details on how this vertex intersection is calculated can be found in section 4.4. The data returned from the server for the vertex intersection only includes edges for students that move through the intersection vertex. This reduces the amount of bandwidth and processing necessary on both the client and server sides. Figure 5 shows an example of highlighting behavior applied to the Sankey Diagram shown in Figure 4 for students who were in the EN (Engineering) college and majoring in EE (Electrical Engineering) during their 5th semester. Additional Sankey Diagrams, as well as higher-resolution versions of Figure 4 and Figure 5 can be found in Appendix A.

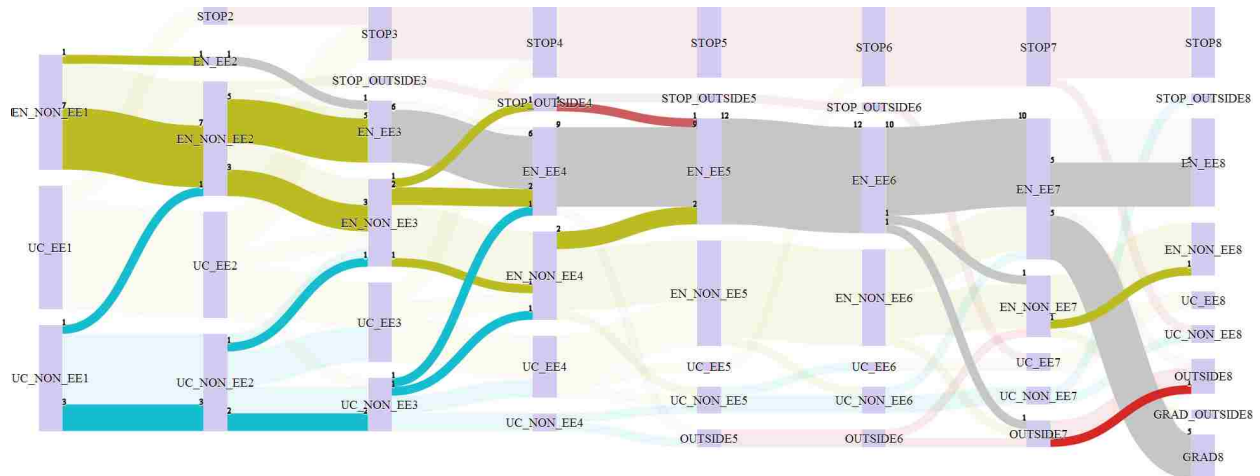


Figure 5 – Sankey Diagram highlighting students who passed through the EN_EE5 for Electrical Engineering students in the 2007 cohort.

In addition to a Sankey Diagram, the client-side application also renders a table showing the number of students in every vertex. The four and six-year graduation rates, along with totals, are displayed as well. In the future, cells within this table could be highlighted when the user is highlighting students who passed through a vertex.

Table 1 – Example output of vertex synopsis on client-side application

	Semester								
Year grad rate								4 year	
Type	semester 1	semester 2	semester 3	semester 4	semester 5	semester 6	semester 7	semester 8	total
STOP	0 (0.0%)	2 (5.1%)	6 (15.4%)	8 (20.5%)	8 (20.5%)	9 (23.1%)	9 (23.1%)	8 (20.5%)	8
STOP_OUTSIDE	0 (0.0%)	0 (0.0%)	1 (2.6%)	2 (5.1%)	1 (2.6%)	1 (2.6%)	0 (0.0%)	1 (2.6%)	1
EN_EE	0	1	7	10	12	12	16	10	10
EN_NON_EE	13	13	10	10	12	11	7	6	6
UC_EE	14	12	9	7	1	1	2	2	2
UC_NON_EE	12	11	6	2	3	3	2	2	2
OUTSIDE	0	0	0	0	2	2	3	4	4
GRAD_OUTSIDE	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	1 (2.6%)	1
GRAD	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	5 (12.8%)	5

4.2. Server Implementation

When the server component is first executed, before allowing any client connections, a JSON configuration information is generated that every client must request before any Sankey Diagrams can be rendered. This configuration information contains a list of the distinct majors and colleges available for each cohort year. It is important to generate this information at startup because there is no guarantee that this data will remain static from cohort-to-cohort. To generate this data, first the database is queried to retrieve a list of the distinct cohorts. Then, for each cohort, the database is queried again for the distinct colleges in that cohort. Finally, for each distinct cohort-college pair, the database is queried to retrieve a list of the distinct majors available. This series of queries is time consuming, and is immutable, which is why it is computed and cached

before allowing any client connections. After this initialization process is completed, the server waits for incoming requests from the client-side application.

For the server-side as a whole, there are two distinct cases for generating queries. In the first case (hereinafter referred to as the “College Group”), the client has specified a college, but the major is unspecified. In the second case (hereinafter referred to as the “Major Group”), the client has specified both a major, a primary college (e.g., Engineering College), and zero or more secondary colleges (e.g., University College). Requests where a major is specified, but no primary college is specified, are considered invalid. For both of these cases, the cohort year must also be specified.

When the server receives a request from the client-side application, the Redis server is checked, using the query string received from the client as the hash key. If Redis has cached the query string response then the cached value is returned immediately without performing any of the query validation performed below because that the value is cached indicates that it has previously passed the query validation. A more detailed explanation of the caching strategy can be found in section 4.5. If the value has not been cached in Redis, the server performs a preliminary validation in order to check that the values are within the expected ranges (e.g., a valid cohort year is one for which student data is available). If any of the following validation fails, then no further processing is done, and an error message is returned to the client. In a College Group case, the server checks the database to ensure that the requested college exists within the cohort year requested. Additionally, for the College Group case, the primary college is the only valid college. In Major Group cases, the list of valid colleges will include both the specific college for the major (e.g., Engineering) and zero or more secondary colleges that also include the major (e.g., University College); for the purposes of the Sankey Diagram, any of these colleges

constitutes a student being in a correct college. Thus, the server checks the database to ensure that the requested major exists within the colleges and cohort year requested. Figure 6 shows the flow chart for this validation process.

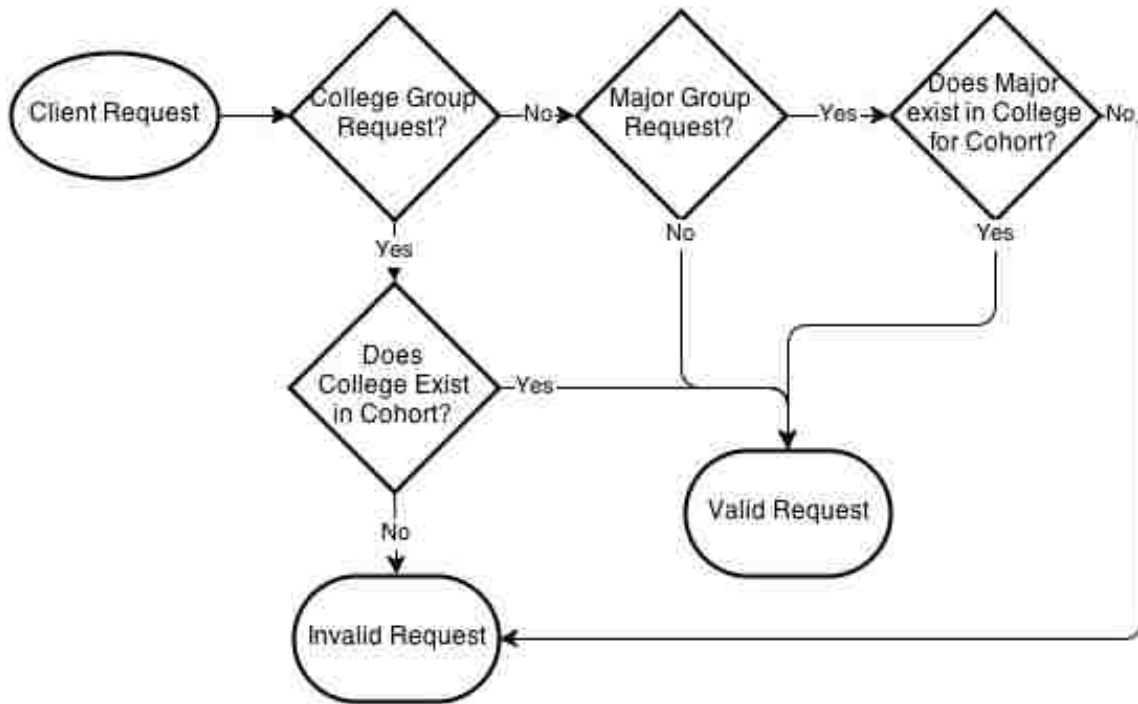


Figure 6 – Client Query Validation block diagram

After performing the client input validation the server checks if additional information is necessary to construct the list of all possible vertices. There are three potential cases. (1) If a user does not request data for a specific college or major, the database is queried to retrieve a list of all distinct colleges within the requested cohort year. (2) For a College Group type request, the

database is queried to retrieve a list of all distinct majors within the college within the requested cohort year. (3) For a Major Group request, no additional database queries are necessary because the no additional information is necessary to create the list of all possible vertices. Figure 7 shows the flow chart for how the server checks if more information is necessary to construct the list of possible vertices.

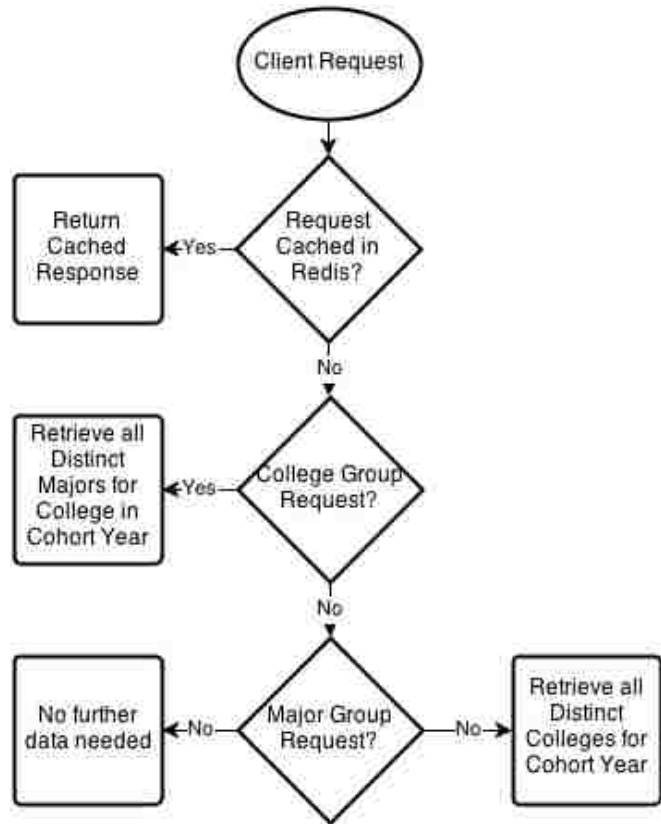


Figure 7 – Flow chart for requesting additional information from database

Once the list of all possible vertices has been constructed, the server builds an SQL query to be sent to the PostgreSQL

database server. These queries are done as a series of sub-queries each of which successively filters the data returned by the previous query. Information regarding the database schema can be found in section 4.3. The first of these queries is used to filter out students who are outside of the selected cohort year and within the selected college and major for a minimum of one semester; additionally, students that do not fit within optional filtering criterion such as gender, ACT score, SAT score, etc. This first query is used as the basis for every type of incoming query, including queries for a vertex intersection. The next set of queries is only run when calculating a vertex intersection. The final query returns each distinct student returned from the previous filtering queries the set of all semesters enrolled, including the students major, college, and whether a degree was obtained.

With the data returned from the SQL query, the server then iterates through every student's sequential list of semesters, building a list of the edges and vertices based on where the current student is for each semester. Every type of request includes the vertices for graduation and dropping out. There are three cases for how the vertices are named: (1) If no college or major is

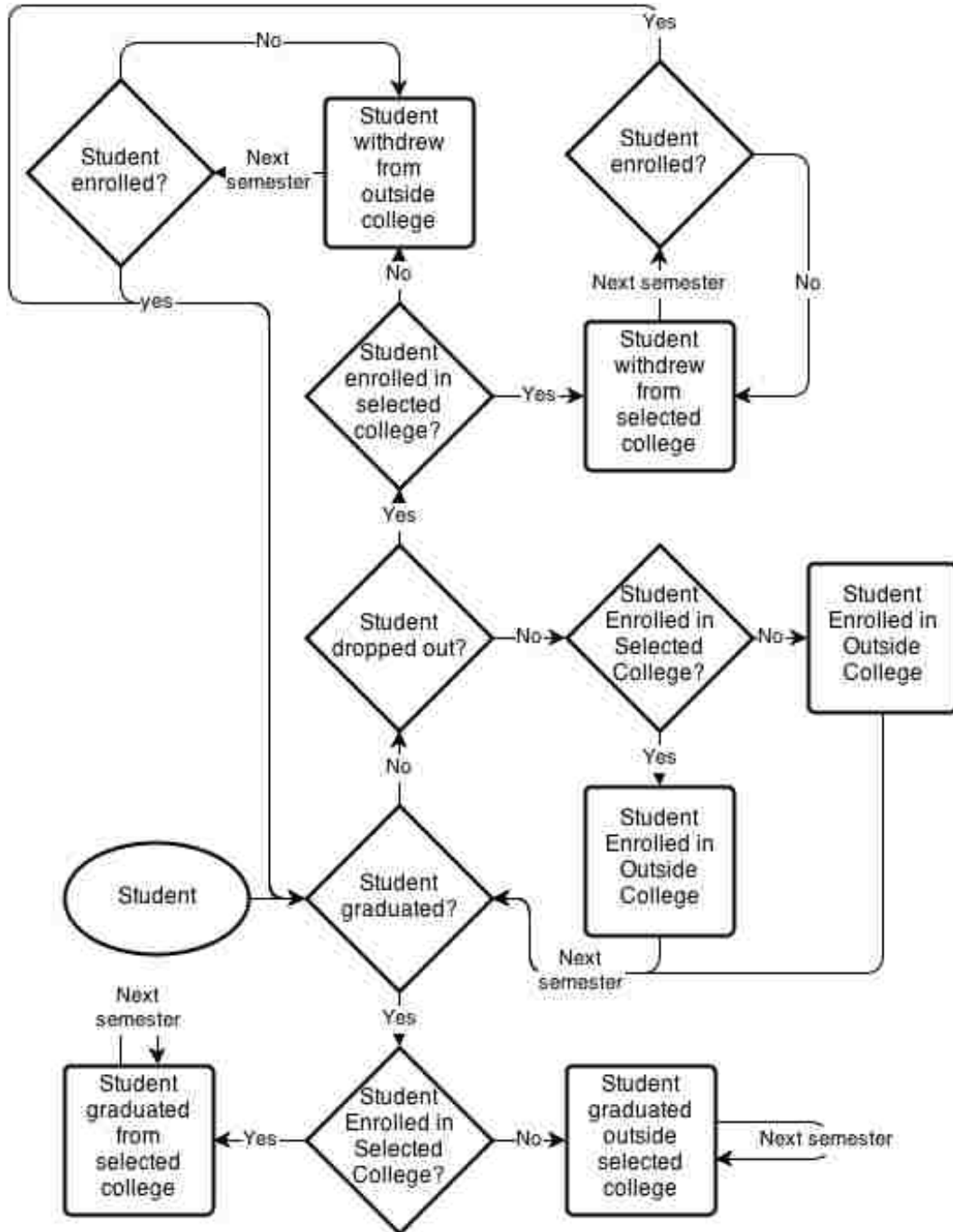


Figure 8 – Flow chart for detecting which flow a student is currently part of for the College Group case

specified, then vertices are created for every college available. (2) In the Major Group case, vertices are created for the college, each of the valid colleges, and for graduating and dropping out outside of the valid major. (3) In the College Group case, vertices are created for every possible major within the college as well as for graduating and dropout out outside of the college. A block diagram for placing students at a particular vertex along a path for the College Group case is in Figure 8. The cardinality (i.e., the weight) of students in each edge is the only state information maintained during the building of the weighted directed graph.

4.3. Server Database Schema

The student data was separated into two tables named *student* and *student_semester*. The *student* table contains pre-institutional factors that remain constant, such as gender, ACT or SAT score, and high school GPA. The *student_semester* table contains data on each semester such as college, major, enrolled status, degree attainment, and GPA. Each semester has a separate row in this table, and has a *student_id* foreign key creating a many-to-one relationship between the *student_semester* and *student* tables.

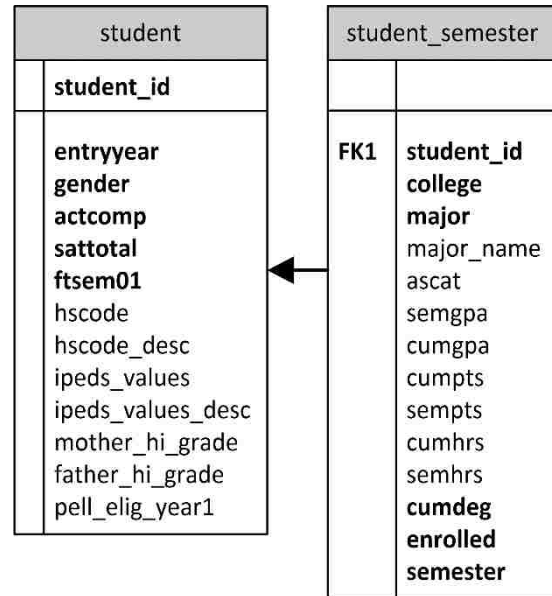


Figure 9 – Database Schema

A visual representation of the database schema can be found in Figure 9.

4.4. Server Handling Client Queries for Vertex Intersection

For a vertex intersection request, the client has requested all of the edges for students who passed through a particular vertex in the Sankey Diagram. As with the non-vertex intersection

requests, there are two distinct cases for generating this kind of query are the College Group and Major Group.

The College Group has three separate cases for the query: (1) student is enrolled in the valid college; (2) student is outside of the valid college; (3) Student has graduated or dropped out. The Major Group has four separate cases for the query: (1) student is enrolled in a valid college and a valid major; (2) student is enrolled in a valid college, but outside of the valid major; (3) student is outside of the valid colleges; (4) student has graduated or dropped out.

Cases where neither a college nor major are specified are general cases of the filtering that occurs with the Major Group. The primary difference is that there are no longer vertices for “*outside*” graduation or dropping out because this special case includes every student regardless of declared college or major.

For case (1) of the College Group, the query includes all students who were enrolled in any valid college for the semester specified by the intersection vertex. For case (2) of the College Group, the query includes all students who were not enrolled in any of the valid colleges for the semester specified by the intersection vertex. For case (3) of the College Group (a special instance of case 2), the query must determine what college a student was enrolled in when they graduated or dropped out. This must be the most recent case, before the semester specified by the intersection vertex, of either graduating or dropping out as both can occur multiple times during a student’s time at a university. If the original client request matches the status of the student (e.g., “grad,” “grad outside,” “drop out,” “drop out outside”), then the student is included in the results.

For case (1) of the Major Group, the query includes all students enrolled in any valid college and the valid major for the semester specified by the intersection vertex. For case (2) of

the Major Group, the query includes all students who were not enrolled in the valid major within any valid college for the semester specified by the intersection vertex. For case (3) of the Major Group, the query includes all students enrolled in any of the valid primary colleges but not in any of the valid secondary colleges (e.g., include Engineering College, but exclude University College) for the semester specified by the intersection vertex. For case (4) of the Major Group (a special instance of case 2), the query must determine what major and college a student was enrolled in when they graduated or dropped out. This must be the most recent case, before the semester specified by the intersection vertex, of either graduating or dropping out as both can occur multiple times during a student's time at a university. If the original client request matches the class status of the student (e.g., graduated or dropped out), then the student is included in the results.

4.5. Server Caching Strategies

Currently, the response for every unique client request (based on the client query string) is cached by the Redis key-value store. The Redis cache is then checked for every incoming client request. The key for the cached response is generated from the query string from the original client request. This caching strategy substantially improves the performance of the

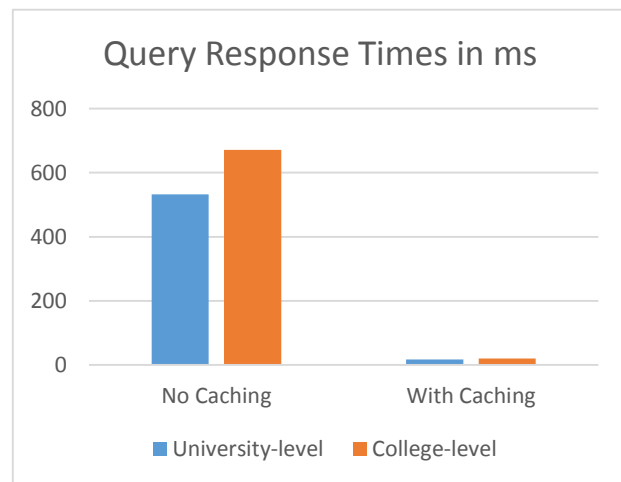


Figure 10 – Query response time comparison

web application, as shown in Figure 10. Caching client responses in Redis improves performance because it replaces the complex series of SQL queries and post-processing on a query result with a single lookup in a key-value store.

The aforementioned caching strategy was adopted because the application is intended to support a cloud-based deployment. This additional efficiency has a direct effect on the cost of running the application in the cloud because it allows cheaper systems to be utilized while maintaining a similar level of performance. The caching strategy also improves responsiveness to the end users, whether or not running the application is running in the cloud or on physical hardware.

Pre-computing and caching the response to every potential request was explored as a potential strategy. However, the overhead of this strategy is substantial. Consider a hypothetical college with six distinct majors over twelve semesters for a given cohort year. Each major has $(6 * 12)$ nodes, and, in addition, the vertex intersection values must be calculated for each of those nodes. This results in $6^2 * 12 = 432$ calculations that must be pre-computed for this single college. This quickly results in a large number of values that must be pre-calculated and stored, making it a poor choice as a caching strategy in regards to both the time and memory necessary.

The caching strategy used here could be optimized with some simple heuristics so that the application's performance is still improved while avoiding caching queries that can be completed quickly. In cases where the client has requested information for building an entire Sankey Diagram, the time for the server to respond to a request can be approximately one to two seconds, and potentially more if the server is under heavy load. This is particularly true in cases where the graph covers a large number of students (e.g., the university-level graph or, to a lesser degree, the college-level graph). These types of queries are the most important to cache in Redis as they have a large effect on the overall performance of the application.

5. Conclusions and Future Work

This thesis described a client-server application that is capable of transforming student cohort data into a weighted, directed, graph, at differing levels of granularity, which is rendered graphically as a Sankey Diagram. Visually representing student data in this manner gives university administration a powerful tool to support attempts to both understand and improve the factors affecting student success that are related to student flows through the university.

The potential exists for attempting to correlate student declared major changes to both academic and non-academic events. For example, there could be courses that an unusually high number of students change their declared major or drop out after taking the course (often colloquially referred to as a “weed-out” course). This correlation could be used by university administration to determine if the course is too difficult or the prerequisite coursework leaves students inadequately prepared. This analysis could be expanded to include some form of automated modeling which predicts student outcomes so that an intervention can be made before the student withdraws from the university [7]. In addition, allowing an analyst to control the visual analytics process by dynamically steering (i.e., choosing various methods of culling the data being shown to allow for a better understanding of what is happening at a university) the creation of the Sankey Diagram [1].

The Sankey Diagrams could be expanded to include changes that occur during the summer session. This additional feature could be used to give a more complete picture of student progression. Adding support for summer sessions introduces complications in displaying the data in a Sankey Diagram as only a subset of enrolled students attend for the summer session. One potential way to solve this problem would be to represent the summer sessions as a sub-flow. That

is, represent the summer session with a vertex that the students pass through only if enrolled for a summer session.

Client-side application accessibility could be improved by adding an option for selecting color-blindness friendly color palettes for the Sankey Diagrams. Color-blindness affects approximately 8.5% of the population—a large enough proportion of the population to make this change worth considering. D3.js includes full support for using ColorBrewer⁸, which has multiple color-blind safe color palettes.

⁸ <http://colorbrewer2.org/>

Appendices

Appendix A	Supplementary Sankey Diagrams	24
------------	-------------------------------------	----

Appendix A Supplementary Sankey Diagrams

In this appendix, a number of additional Sankey Diagrams are presented in order to demonstrate the versatility of the application.

In order to create these diagrams interactively, please visit <http://provostcloud.unm.edu> and select Student Flows Sankey Diagrams.

A.1 All students in 2007 cohort (university-level view)

Figure 11 shows all students in the 2007 cohort. Note how the university-level view shows every available college within the university, and the *GRAD_OUTSIDE* and *STOP_OUTSIDE* vertices are not present.

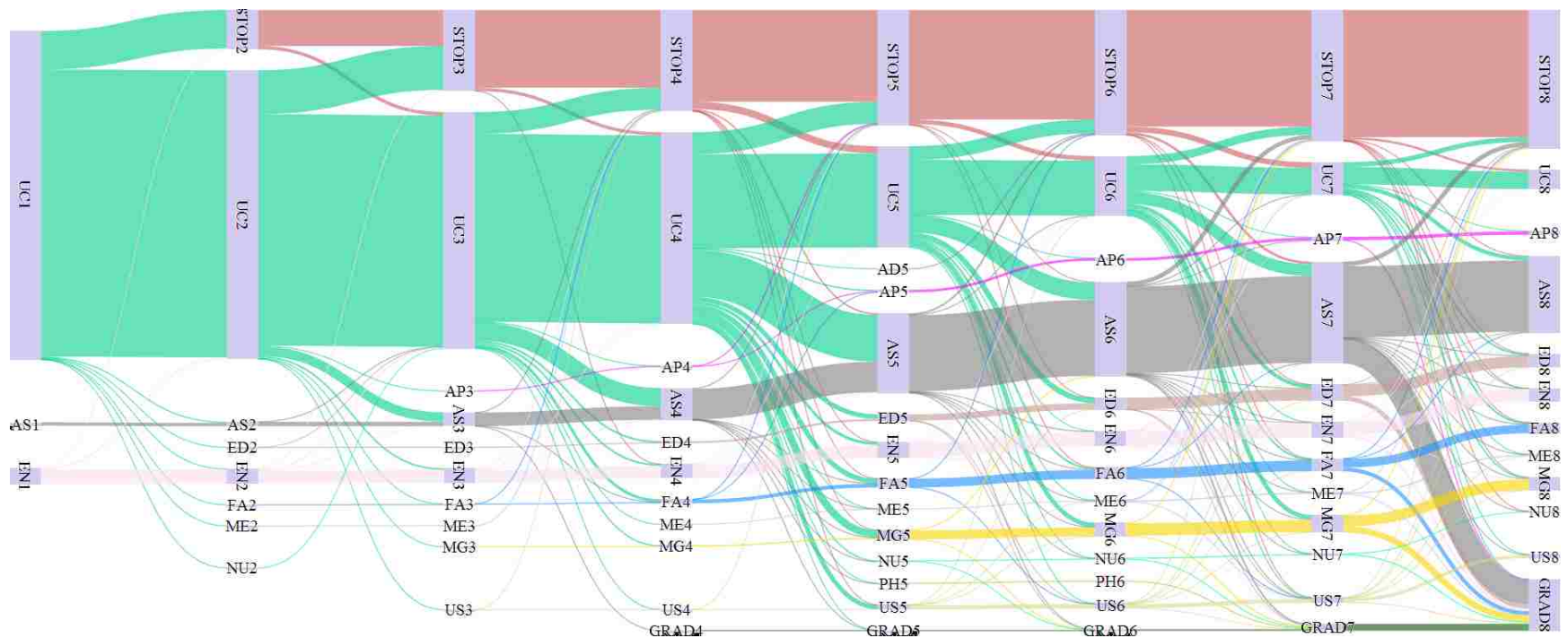


Figure 11 – Sankey Diagram for all students in the 2007 cohort.

Figure 12 shows highlighting applied to Figure 11 for all students in the 2007 cohort who were enrolled in the AS (Arts & Sciences) college during their 5th semester.

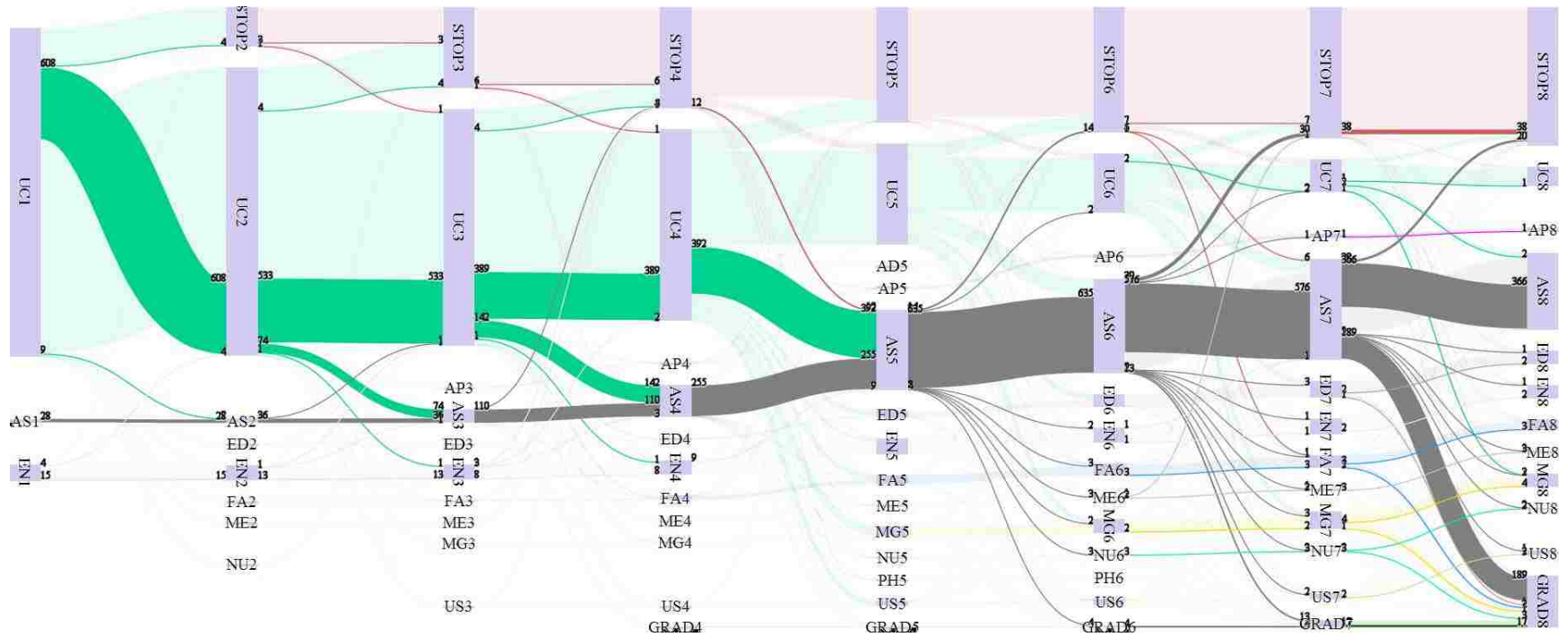


Figure 12 – Sankey Diagram highlighting students who passed through the AS5 vertex for all students in the 2007 cohort.

A.2 College of Engineering students in 2007 cohort (college-level view)

Figure 13 shows all students in the 2007 cohort. Note how the college-level view shows every available major within the college, and the *GRAD_OUTSIDE* and *STOP_OUTSIDE* vertices are present.

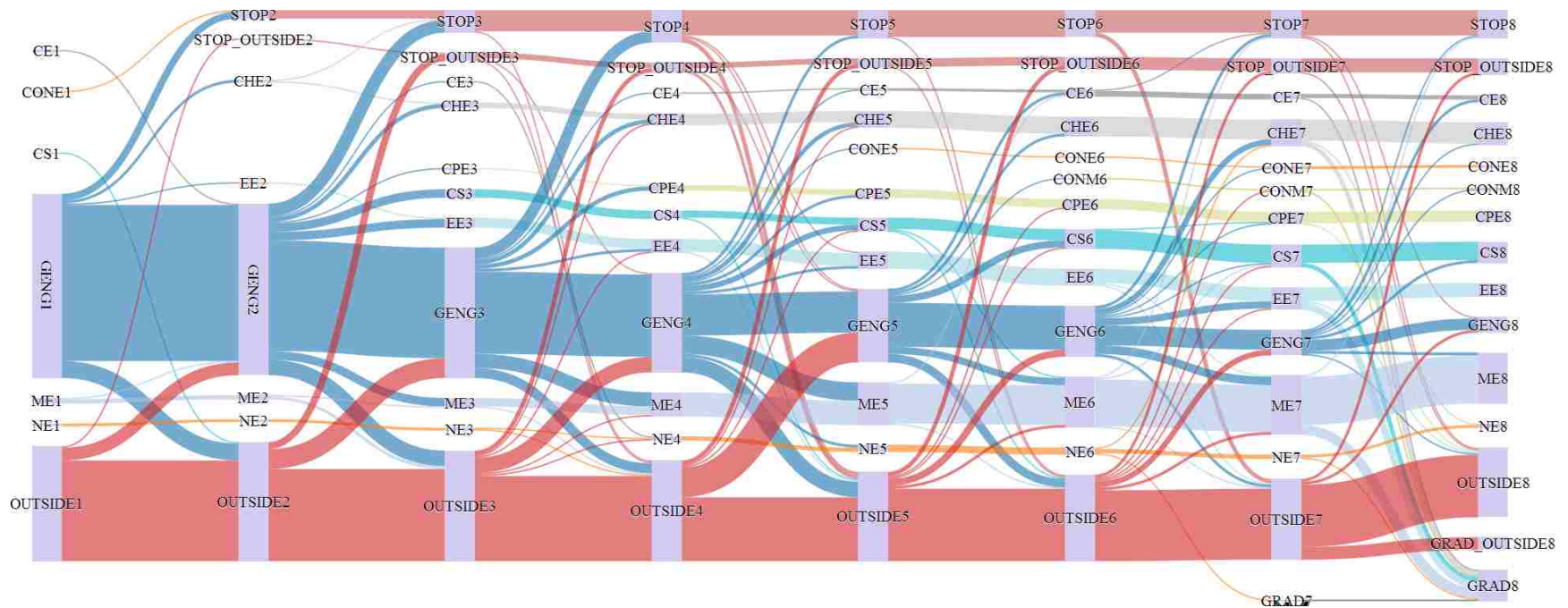


Figure 13 – Sankey Diagram for engineering students in the 2007 cohort.

Figure 14 shows highlighting applied to Figure 13 for all students in the 2007 cohort who were enrolled in the College of Engineering and majored in GENG (General Engineering) during their 5th semester.

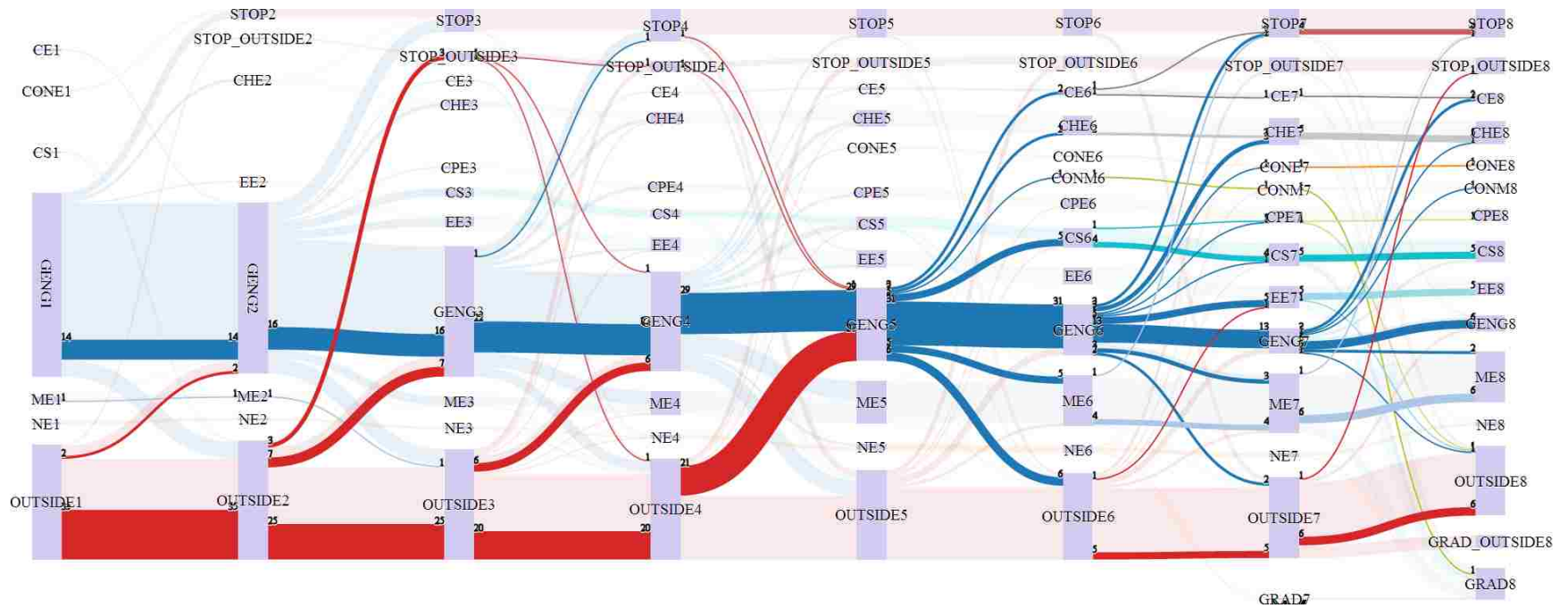


Figure 14 – Sankey Diagram highlighting students who passed through the GENG5 vertex for engineering students in the 2007 cohort.

A.3 English students in 2007 cohort (major-level view)

Figure 15 shows all students who majored in English in the college of Arts & Sciences for a minimum of one semester. Note how the major-level view shows the secondary UC (University College) as well as the primary AS (Arts & Sciences) college, and the *GRAD_OUTSIDE* and *STOP_OUTSIDE* vertices are present.

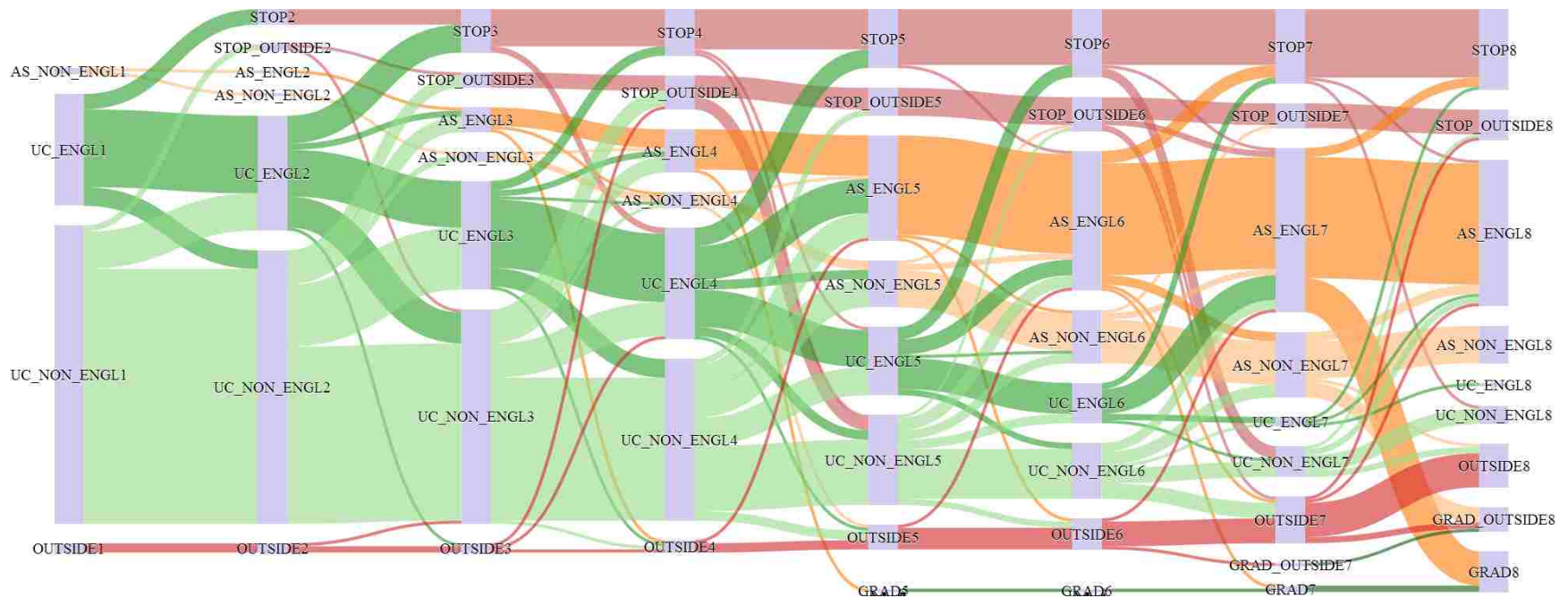


Figure 15 – Sankey Diagram for English students in the 2007 cohort.

Figure 16 shows highlighting applied to Figure 15 for all students in the 2007 cohort who were enrolled in the College of Arts & Sciences but were in a major other than ENGL (English) during their 5th semester.

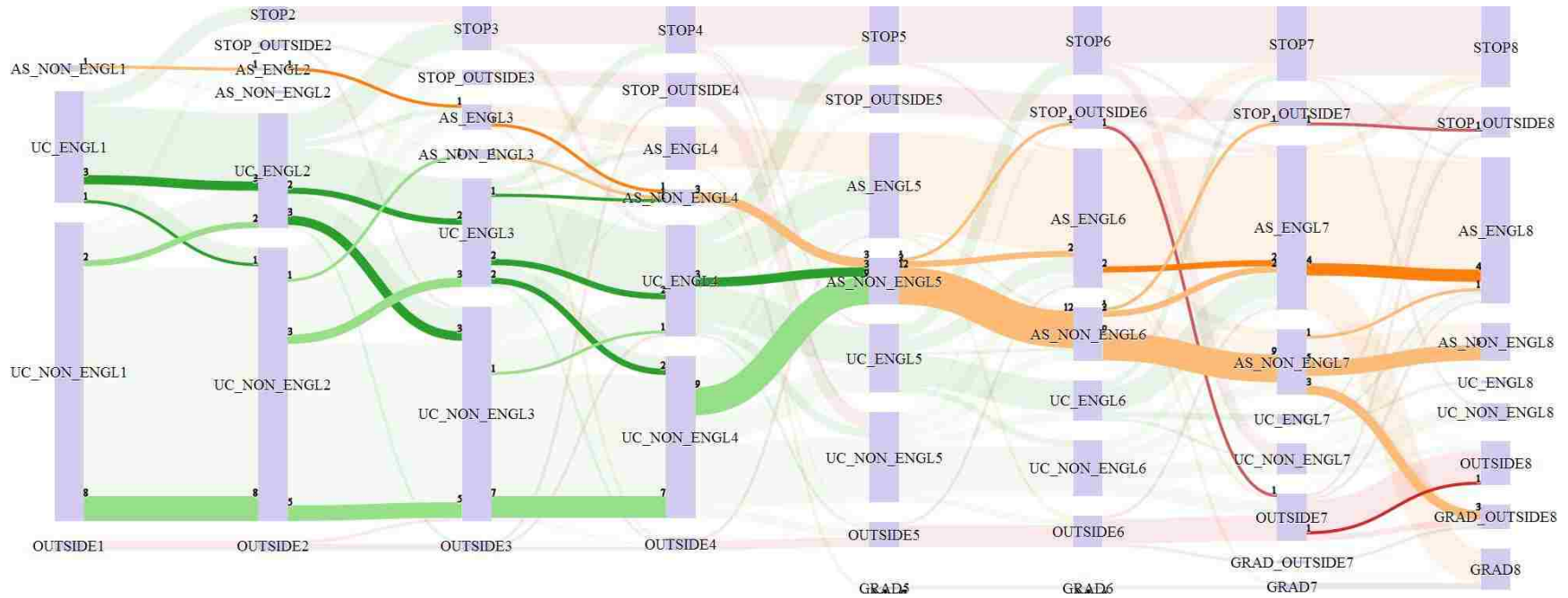


Figure 16 – Sankey Diagram highlighting students who passed through the AS_NON_ENGL5 vertex for English students in the 2007 cohort.

Figure 17 shows highlighting applied to Figure 15 for all students in the 2007 cohort who were enrolled in the College of Arts & Sciences and majored in ENGL (English) during their 5th semester.

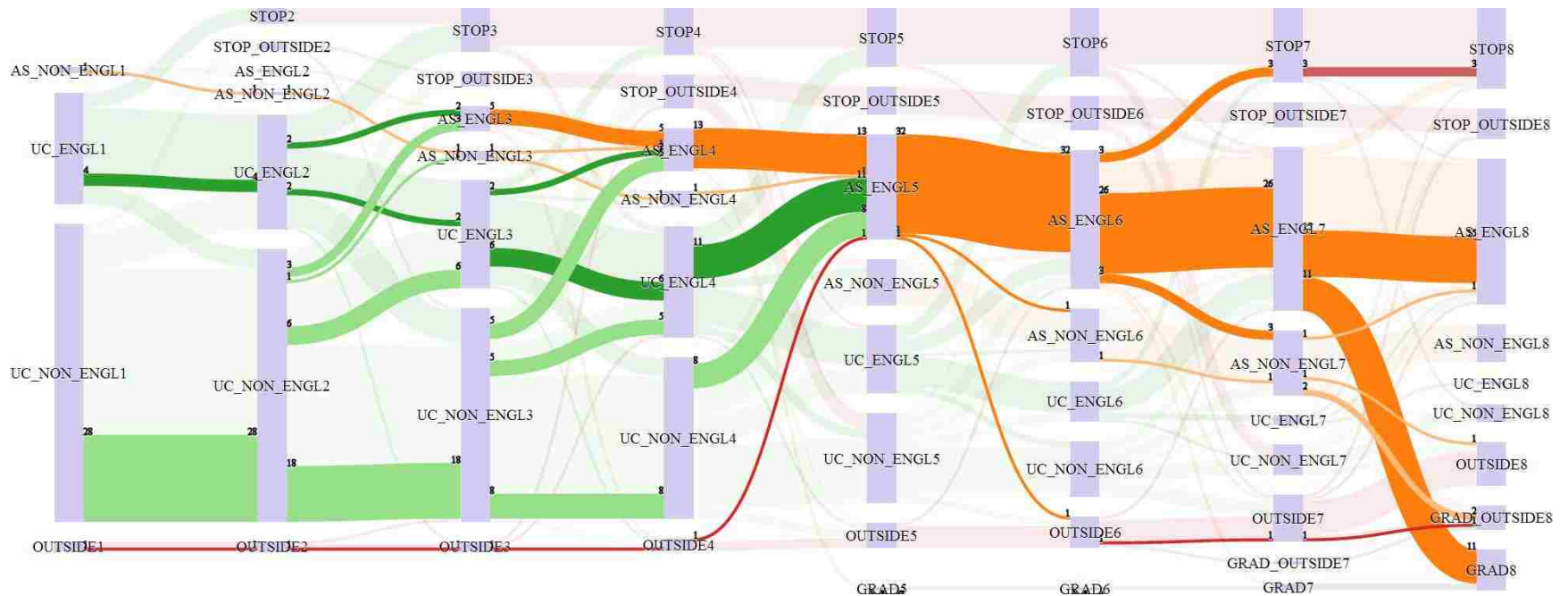


Figure 17 – Sankey Diagram highlighting students who passed through the AS_ENGL5 vertex for English students in the 2007 cohort.

A.4 Electrical Engineering students in 2007 cohort (major-level view)

Figure 18 shows all students who majored in Electrical Engineering in the College of Engineering for a minimum of one semester. Note how the major-level view shows the secondary UC (University College) as well as the primary EN (Engineering) college, and the *GRAD_OUTSIDE* and *STOP_OUTSIDE* vertices are present.

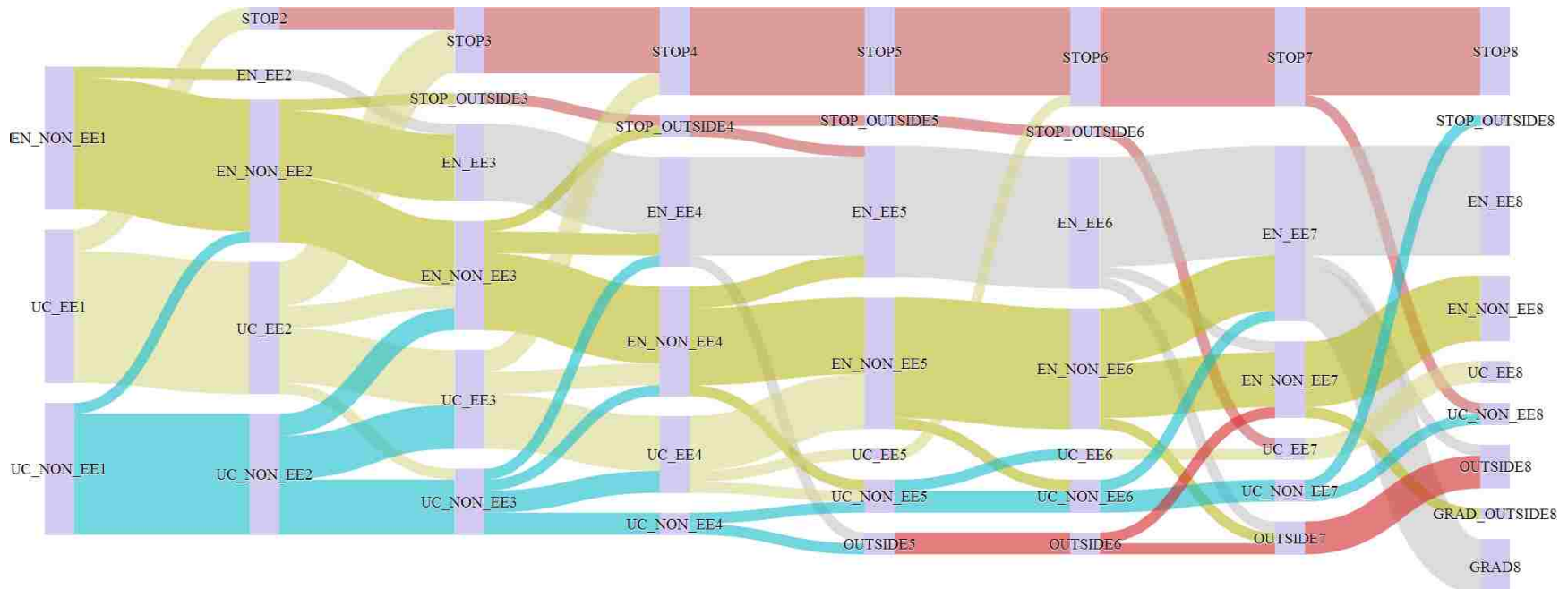


Figure 18 – Sankey Diagram for Electrical Engineering students in the 2007 cohort.

Figure 19 shows highlighting applied to Figure 18 for all students in the 2007 cohort who were enrolled in the College of Engineering but were in a major other than EE (Electrical Engineering) during their 5th semester.

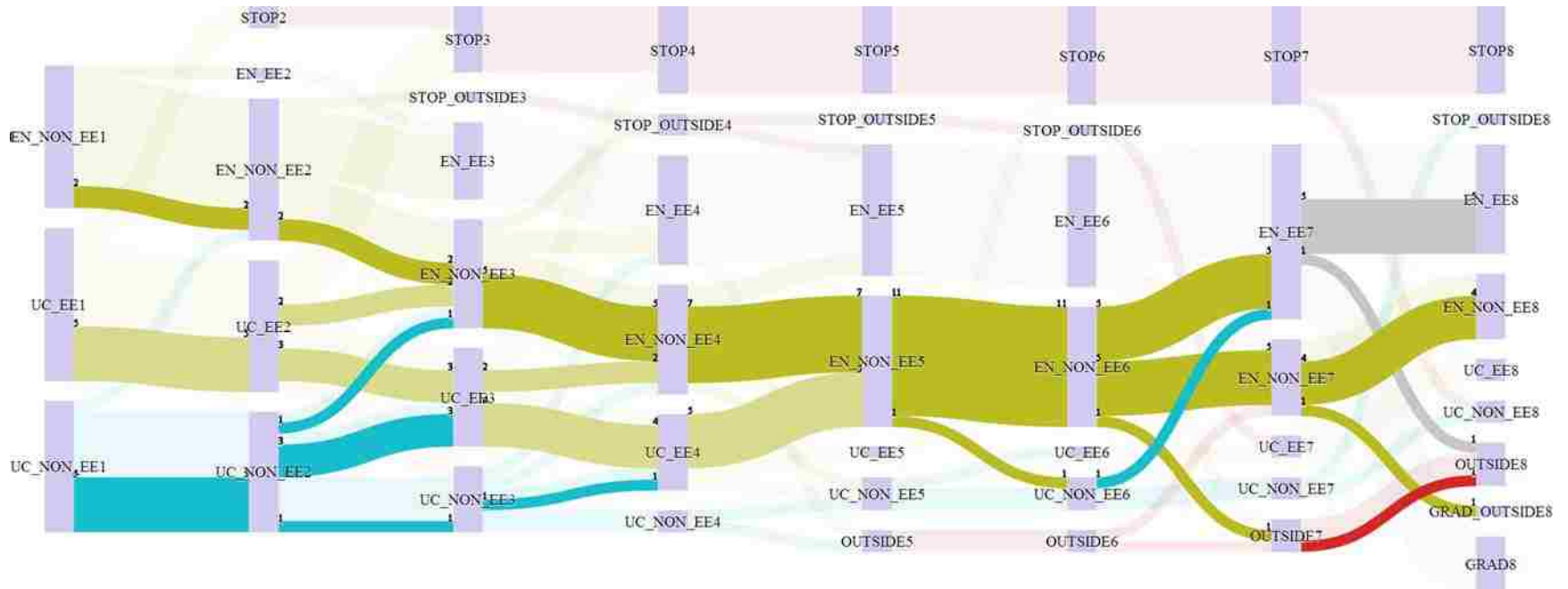


Figure 19 – Sankey Diagram highlighting students who passed through the EN_NON_EE5 vertex for Electrical Engineering students in the 2007 cohort.

Figure 20 shows highlighting applied to Figure 18 for all students in the 2007 cohort who were enrolled in the College of Engineering and majored in EE (Electrical Engineering) during their 5th semester.

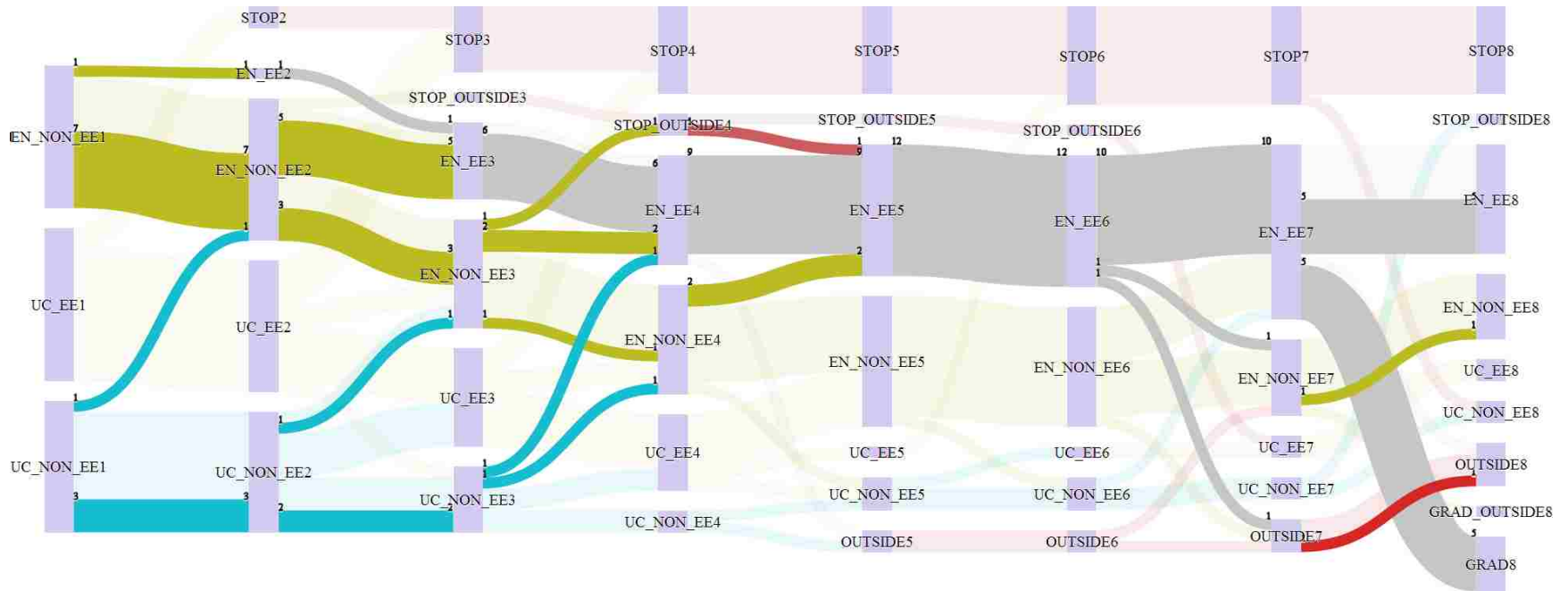


Figure 20 – Sankey Diagram highlighting students who passed through the EN_EE5 vertex for Electrical Engineering students in the 2007 cohort.

References

- [1] J.-D. Fekete, "Visual Analytics Infrastructures: From Data Management to Exploration," *Computer*, pp. 22-29, July 2013.
- [2] M. Schmidt, "Der Einsatz von Sankey-Diagrammen im Stoffstrommanagement," *Beitraege der Hochschule Pforzheim*, no. Nr. 124, 2006.
- [3] P. Riehmann, M. Hanfler and B. Froehlich, "Interactive Sankey Diagrams," in *Information Visualization*, 2005.
- [4] M. Bostock, V. Ogievetsky and J. Heer, "D3: Data-Driven Documents," *IEEE Transactions in Visualization and Computer Graphics*, vol. 17, no. 6, pp. 2301-2309, 2011.
- [5] B. Noel and S. Moore, "Gauss-Seidel Method," [Online]. Available: <http://mathworld.wolfram.com/Gauss-SeidelMethod.html>. [Accessed 19 10 2013].
- [6] M. M. Strout, L. Carter, J. Ferrante, J. Freeman and B. Kreaseck, "Combining Performance Aspects of Irregular Gauss-Seidel via Sparse Tiling," in *Workshop on Languages and Compilers for Parallel Computing*, College Park, MD, 2002.
- [7] P. Suhem, Z. Zahid and F. Merchant, "Application of Data Mining in Educational Databases for Predicting Academic Trends and Patterns," in *Technology Enhanced Education*, 2012.
- [8] P. Baepler and C. J. Murdoch, "Academic Analytics and Data Mining in Higher Education," *Internation Journal for the Scholarship of Teaching and Learning*, vol. 4, no. 2, July 2010.