Analysis of Student Coenrollment Networks

Kar Epker [*]

April 8, 2015

Abstract

In this thesis, we investigate student-student interactions through a well-defined student social network resulting from coenrollment in classes. Starting from the idea that a liberal arts education aims to connect students to a variety of ideas and perspectives, we look at the interactions that students have in the classes they take to try and determine their degree of exposure to different communities than those with which they typically interact. As a preliminary result, we propose one metric to measure the diversity of classes a student takes. We give a few case studies to show that this metric does capture the extent to which a student has been exposed to a diverse set of classes compared to students in similar contexts. Ultimately, we wish to establish a metric to measure the intellectual diversity of a student's undergraduate education.

1 Introduction and Motivation

1.1 Structure of this thesis

This thesis is split into four sections. In Section 1, the Introduction section, we look into the motivation for building student social networks based on coenrollment and what we aim to measure from these networks. We also define the structure of the networks and provide necessary technical and mathematical background for the rest of this thesis.

In Section 2, the Implementation section, we look at the structure of the code used to facilitate the analysis. Specifically, we explore how the networks were built and stored, and how the analysis was done.

In Section 3, the Analysis section, we survey initial metrics both to better understand the population of students and their connections as well as to understand how the information gathered can be used to assess richness of student curriculum and exposure to diverse groups and ideas. We look specifically at one metric across the entire population of students in our data set as well as a few case studies.

[*]Advisers: Tim McKay, Rada Mihalcea

Finally, in Section 4, the Results and Conclusion section, we formally propose a metric to measure the connectedness of a student to various people, and by proxy, ideas, during their undergraduate career.

1.2 Motivation

A student's experience at University is often reduced to a degree and perhaps a transcript listing courses they take and grades they receive. The transcript is usually reduced to a single number, a grade point average (GPA), which can be used as a heuristic that roughly corresponds to the student's academic performance. Within the school, the GPA is used as a basis of determining which honors and awards a student might receive upon graduation [32]. Beyond school, employers may give job interview preferences to students with high GPAs, and graduate programs use GPA to determine who they admit.

As a heuristic, this thesis does not question the usefulness of the GPA or its purpose. However, we wish to question the overwhelming attention GPA has received as a marker for student success. GPA captures students' performances in class as a proxy for their acquisition of knowledge, but knowledge can be acquired through many media; one need not attend University to learn. The availability of information on the Internet shows that knowledge has moved outside the ivory tower, and beyond that, the rise of Massive Open Online Courses (MOOCs) allow digital students to receive recognition from accredited universities. Therefore, the purpose of attending a University can not only be for knowledge. We argue that two more dimensions define a student's experience, especially during their undergraduate years: the University allows students to build a social and professional network, and also gives them the opportunities to view different perspectives. This last point is essential to the liberal arts education, which strives to expose students to diverse views through distribution requirements.

That said, even students pursuing areas of study not typically associated with liberal-arts education, such as science, technology, engineering, and math (STEM) may benefit from exposure to diverse intellectual ideas. Regardless of degree or career path, students will likely be working among peers after they graduate, who will come from different backgrounds and bring different ideas. It's essential that students are prepared to consider and embrace views that surprise and challenge those they hold.

While we may continue to make arguments on the importance of taking an intellectual rich course schedule as an undergraduate, it ultimately falls outside the scope of this thesis. In this thesis, we take as a postulate that exposure and connection to different groups is an essential part of university experience and contributes to a student's mental and emotional growth. Our goal is to begin the process of quantifying that exposure based on the various classes students elect to take over their undergraduate years.

Our research will take analyses that have been done on social networks and apply them to a network of students. We will construct this student network based on transcripts, taking students as vertices and using coenrollment data to draw edges between students who share this intellectual connection. This network of students can be considered a social network, because it represents how students are connected through the courses that they take. However, unlike other social networks that have been studied in the academic community, we will use administrative data to weight this network as opposed to results of surveys [30].

Our preliminary analysis of this network of students will mostly focus on the structural aspects of the network and how the qualitative aspects of the network can be quantified through its structure. Can we find certain groups that are more connected than others? Do certain groups have more interaction than we might expect from a random graph? What kind of quantifiable measures might indicate that a student is more connected to various groups of students than their peers.

In the rest of this introduction, we explore previous research on social networks, and then provide a technical background for the computational part of building and assessing the networks. We finish the introduction by more rigorously defining the social networks. The next sections delve into the computational infrastructure required for building and analyzing these networks, then analysis on the networks and results.

1.3 Previous Research

Much research has been done on social networks: their evolution over time [2, 17], their effect on other aspects of life [6, 9, 13], and how to quantify their structure [5, 8, 11, 12, 24]. These studies often focus on online social networks (e.g. Facebook, Twitter) [6, 9, 24], social networks created by citations [4, 15], or even the social networks of other animals [16, 19, 20].

When it comes to the academic side of networks, many studies focus on the social benefits of networking, especially for minority students [25, 30]. Beyond students, some research investigates the social benefits of networking for entire institutions [22]. Most of these studies approach networking from a sociological perspective: networks must be built, as opposed to examining networks that already exist. Additionally, many studies of networks are small and rely on observation [24, 30], where students or researchers record subjective data about strength of connections without any rigorous quantitative method.

This social approach to networking extends to studies on student interactions. Many of these are educationally focused. Some give instructions on how to structure student-student interaction to help the students learn [26]. Many focus on student-teacher interaction, noting that the quality of this determines students attitudes toward school and the learning process [3].

Not much attention has been given to the type of research that we wish to complete: a large-scale quantitative investigation of a well-defined student social network.

1.4 University context and data background

To place the data we will use in context, it is useful to consider some demographic information about the University of Michigan. The University of Michigan is a large, public, Midwestern research institution. It is split into several undergraduate and graduate schools and programs. A majority of undergraduates, about 61%, are enrolled in the College of Literature, Science, and the Arts (LSA), which contains humanities, math, and natural and social sciences courses [33]. The second largest school is the College of Engineering, which contains students wishing to concentrate in engineering disciplines, such as mechanical engineering, civil and environmental engineering, aerospace engineering, electrical engineering, etc. Additionally, several smaller schools, for example, the School of Music, Theatre & Dance and the School of Nursing, exist for other specialized conentrations. In total there are 18 schools represented in our data.

It should also be noted that several programs do not allow students to apply until they have reached a certain class standing in another school. For example, the Gerald R. Ford School of Public Policy does not accept students before they reach junior standing. The Stephen M. Ross School of Business requires students to apply their freshman year. Obviously, some exceptions exist to these rules, the Ross School has a preferred admission program that guarantees incoming freshmen a spot. Additionally, it is possible to transfer between schools at the University. I have most commonly heard of students transferring between LSA and Engineering, but nothing bars students from applying into other programs at the University as well. Keeping this in mind is important as background information about the environment in which students build their social networks. Some programs are very small or have very specific distribution requirements, which impacts who students in those programs encounter.

Degrees and, in particular, majors are each offered by one specific school. Civil Engineers, for example, must receive their B.S.E. through the College of Engineering, whereas Classics majors must receive their B.A. through LSA. Of course, exceptions exist to this as well. Of particular interest to me is Computer Science, which is offered through both LSA and Engineering, though courses are all taught by the Electrical Engineering and Computer Science department, which resides in the College of Engineering.

Beyond the split of students into schools at the University, it is useful to consider additional demographic information. A majority (59%) of the University's undergraduate students come from the state of Michigan. 66% of these students are white, followed by smaller populations of Asian (13%), Hispanic (5.14%), African American or Black (4.63%), and Native American (0.21%) students. The gender enrollment for the overall University is roughly half men and half women (52% and 47%, respectively), though these proportions can change quite a bit based on school. Engineering, for example, is infamous for being male-dominated (75%), and computer science, my discipline, even more so (82%) [1, 33].

With this background in mind, we may now consider the data set we were provided for this thesis. Because universities are beginning to pay more attention to how data can help them improve their learning process and understand the environment in which students learn,

Table 1: Enrollment information included in the data set

Field	Demographic information and notes
Gender	Male: 50.4%, Female: 49.6%
Ethnicity	Hispanic: 4.1%, American Indian or Alaska Native: 0.2%, Asian only: 11.1%, Black or African American: 4.2%, Hawaiian or other Pacific Islander: $< 0.1\%$, White only: 63.1%, Unknown: 8.9%, Nonresident Alien: 8.3%
First term	2006: 10.1%, 2007: 12.4%, 2008: 12.0%, 2009: 12.7%, 2010: 13.5%, 2011: 13.0%, 2012: 13.0%, 2013: 13.2%. Note that actual terms are broken out in the data set, but for brevity we include only the year here.
Transfer status	Transfer: 14.1%, Non-transfer: 85.9%
Major 1	One of a set of codes generalized across several Universities participating in learning analytics. NA if no major is on file. For brevity, we do not include demographic information here.
Major 2	The same format as Major 1.
College	18 programs are included, including: Architecture: 0.8%, Art: 1.8%, Business Administration: 0.1%, Dental Hygiene: 0.1%, Education: 0.3%, Engineering: 12.0%, Kinesiology: 0.9%, LSA: 70.6%, Music, Theatre, & Dance: 1.0%, Natural Resource and Environment: 0.1%, Nursing: 1.6%, Pharmacy: 0.2%, Public Policy: 0.1%. 9.7% of students are not in any degree program, and are marked as "NA".

the data from which we build the networks were already available as part of efforts of the University of Michigan Provost's Learning Analytics Task Force.

The data were split into two tables. One included information on when students took the courses offered (described by Table 1), while the other included more details on the students themselves (described by Table 2),

1.5 Background on Graphs

Before beginning this section, I must acknowledge the indispensibility of Mark Newman's textbook *Networks: An Introduction*, which provides a broad overview of various topics of

Table 2: Enrollment information included in the data set

Field	Description
ID	A pseudonymized ID of a student who took the course.
Subject	The subject in which the student is enrolled. These departments are the names of departments in the University of Michigan.
Course Code	The "level" of a course, for example, 101.
Course Credit	The number of credits a student receives upon completing the course.
Term	The term in which the student took the course.

study in networks, from mathematical results in graph theory, to algorithms, to concrete applications. We will refer to this book several times over the next few sections.

We start with the mathematical definition of a graph. A graph is defined as $G = (V, E)$, where V is a set of vertices in the graph and E a set of edges. For the sake of this thesis, network and graph are synonyms, and will be used interchangeably. Among the community, while the terms seem to be used differently depending on the circumstances, no accepted rigorous distinction exists between them. In fact, some define them as synonyms [23].

Graphs can be directed or undirected; that is, edges between vertices can point from one vertex to another, or they can have no orientation and simply connect the vertices. Mixed graphs may have both directed and undirected edges.

Additionally, graphs may be weighted or unweighted. Weighted graphs have a function $w : E \to \mathbb{R}$ such that every edge of the graph has a specific weight associated with it.

Graphs may meet certain useful constraints on the types of edges they have. "Loops" in a graph are edges from a vertex to itself. Some graphs may meet the constraint that every pair of vertices defining an edge is distinct. This means that there may only be at most one edge between any given pair of vertices. Graphs that do not have loops and have distinct edges are called "simple" graphs. Graphs that do not meet these criteria are called multigraphs. If a path of edges exists from every vertex to every other vertex in the graph, a graph is called "connected".

Graphs may also be further classified if they meet certain structural criteria. For example, a tree is a connected, undirected graph that contains no closed cycles [23]. A cycle is a path from a vertex to itself. An analogous construction for directed, connected graphs is called an "acyclic directed network", which has no cycles and the interesting property that it can be drawn with all edges pointing downward.

More abstract representations of networks exist as well. For example, a hypergraph is a network in which edges may connect more than two vertices [23]. Newman gives the example of edges representing films connecting a network of actors [23]. Because more than two actors may appear in films, edges must be able to connect an arbitrary number of actors. Another way to represent this hypergrpah is using a bipartite network. A bipartite network allows connections only between vertices of unlike types. For example, consider the hypergraph of movies connecting actors. A bipartite version of this network would allow actors only to be connected to movies in which they have appeared.

Hypergraphs and bipartite networks certainly have the ability to depict information that cannot be captured by a network in which all the vertices are of the same type. These networks do not appear in this thesis, for reasons that will be explored in the next section.

1.6 Representing graphs on a computer

Because graphs may take so many forms, they can be used to represent several important problems in computer science. One of the most famous problems in computer science is the "Traveling Salesperson Problem" or TSP, which asks for the path with the smallest weight that visits all the vertices of a given graph. Beyond their usefulness in representing certain tricky algorithmic questions, one may use graphs to ask questions about concrete relationships as well. For example, how many papers has a pair of researchers authored together? Which pages on the Internet link to other pages on the Internet? Which students have coenrolled in a course?

Due to their wide use, representing graphs and executing algorithms on them has become an important part of undergraduate computer science curricula. I summarize some important points from my undergraduate education on graphs that served as background for this research.

For the rest of this thesis, let the number of vertices $|V| = n$ and the number of edges $|E| = m$.

Two structures are usually employed when representing graphs on a computer.

- **The adjacency list:** The adjacency list is essentially a "list of lists". Every element in an adjacency list corresponds to a vertex v_k in a graph. Each of these vertices v_k points to a list of neighboring $v_i, v_j, \ldots$. The presence of v_i in v_k's neighbor list implies that the edge $(v_k,\ v_i)$ exists. Note that if the graph is undirected, the edges (v_k, v_i) and (v_i, v_k) will both appear. If the graph is weighted, the neighbor list may store a pair consisting of a vertex and the corresponding edge weight.
- **The adjacency matrix:** The adjacency matrix is essentially a square table with row and column headings corresponding to vertices. Every cell has an address (v_k, v_i), and the entry in that cell denotes whether v_k and v_i share an edge and the edge's weight. Note that for undirected graphs, this matrix is symmetric across the diagonal.

Newman's book summarizes the asymptotic runtime of four common operations on each of the data structures in a convenient table. I duplicate this in Table 3 and also add a space entry to further elucidate the tradeoffs each structure has. The four common operations are:

- **Insert**: Insert a new edge into the network.
- **Delete**: Delete an edge from the network.
- **Find**: Testing whether two vertices are connected by an edge.
- **Enumerate**: List all neighbors of a given vertex.

Note that Table 3 does make some assumptions. It assumes that the adjacency matrix and adjacency list are stored in a structures that allow random access. It also assumes that the neighbors list for every item in the adjacency list is stored in a random access structure and is unsorted. This assumption is not always correct. Implementations of adjacency lists often decide to keep the neighboring lists in a different type of data structure, which changes the algorithmic complexity of the operations. For example, if the neighboring list were stored in a binary search tree, insert, delete, and find would cost $O(\log m/n)$ time.

Operation	Adjacency Matrix	Adjacency List
Insert	$O(1)$	$O(1)$
Delete	$O(1)$	$O(m/n)$
Find	$O(1)$	$O(m/n)$
Enumerate	$O(n)$	$O(m/n)$
Property	Adjacency Matrix	Adjacency List
Space	$O(n^2)$	$O(m)$

Table 3: Asymptotic complexity for various operations on adjacency matrices and lists. Courtesy *Networks: An Introduction* [23].

Before we further unpack the information in Table 3, it is useful to consider one more classification of graph that computer scientists find particularly useful: sparse and dense. Sparse graphs have a number of edges approximately the same as the number of vertices, that is $m \sim n$. Dense graphs have a number of edges approximately equal to the square of the number of vertices, that is $m \sim n^2$. It's useful to note that for a simple graph, the maximum number of edges is:

$$\frac{n(n-1)}{2} \in O(n^2)$$

Thus the approximation that dense graphs have edges approximately equal to the square of the number of vertices.

Adjacency lists are particularly useful when graphs are sparse. Having a number of edges approximately equal to the number of vertices means that, on average, every v_k has one neighbor if the graph is directed, or two if the graph is undirected. Either way, the adjacency list takes up only $O(n)$ space for a sparse graph. An adjacency matrix, on the other hand, takes up $O(n^2)$ space no matter the size of m, making the adjacency list a better choice for a sparse graph.

For dense graphs, in terms of space, the choice is less clear: both structural representations require $O(n^2)$ space. However, when we examine the time complexity for the four common operations listed in Table 3, it is clear that the adjacency matrix representation of a graph offers better performance.

To return to a point made in Section 1.5, after examining these structures, it becomes clear why hypergraphs and bipartite graphs are less useful. For computer scientists and researchers, it is much easier to use existing libraries than to create them. Because the adjacency matrix and adjacency list are so well-known, many libraries that use these implementations exist [23]. On the other hand, hypergraphs are not as often used. Not even Newman explores structures that might be able to represent them [23]. Bipartite graphs could be hacked into an adjacency matrix or list representations, but doing so loses the additional invariants they impose on their vertices and edges, and is likely to wreak havoc with the type system in any strongly-typed language such as C++. Any library that provides an encapsulation of a graph would likely be broken due to the fact that they are not designed to handle this type of structure. Developing a library to represent hypergraphs actually may be a good thesis project, but it is beyond the scope of this one.

1.7 The networks we'll study

We will consider a few different networks, represented by graphs, to try and extract information. First, let's describe the information we want out of the networks. Then, we'll consider ways to represent them.

Our overall goal in this thesis is to be able to determine how connected students are based on coenrollment; that is, which other students are in their classes? Is this community of students homogenous, or does it consist of a mix of students in the University community?

This idea of students connected to other students actually closely mirrors the example we gave in Section 1.5 of actors connected to other actors based on the films they appear in. The network we want, then, is a truly bipartite network. Unfortunately, for reasons that we discussed in Section 1.6, no libraries exist that provide good representations of bipartite graphs.

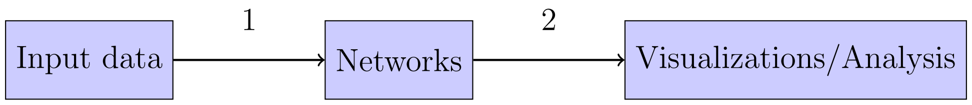
It is useful, then, to consider ways we may be able to represent a bipartite graph that current libraries implement. Newman explores the one-mode projection, which is sufficient for our purposes [23]. Consider our bipartite network: vertices representing students connected to vertices representing classes. To form the one-mode project from this two-mode network, we add edges between the students who have a class in common. Unfortunately, this representation loses information on the number of classes the students share in common. To reduce this loss of information, Newman suggests weighting the edges, in our case, by the number of classes in common [23]. This weighting still loses some information: it doesn't record the number of classes or in which classes students were coenrolled. However, we can solve this problem by simply storing a list of students and the classes they have taken and looking coenrollment data up in this list if we need it. This solution sacrifices space complexity—we no longer store the vertices corresponding to classes in our graph—for time complexity.

We can make other improvements to the weighting between students in our one-mode graph projection. While simply knowing the number of classes in which students were coenrolled is useful, not all coenrollment implies an equal intellectual connection. Coenrollment in a large introductory lecture is different from coenrollment in a small upper level seminar class. Additionally, coenrollment in a class that meets four times a week is different from coenrollment in a class that meets once a week for half a semester. With this in mind, we may weight edges between students with respect to these variables. Our weighting formula for the edge between students v_i and v_j is given by (1).

$$w(v_i, v_j) = \sum_{c \in C} \frac{H_c}{|E_c|} \tag{1}$$

where C is the set of courses the students are coenrolled in, H_c is the number of credit hours for a specific course c, and E_c is the set of students enrolled in the course. While we could choose to represent multiple cases of coenrollment between v_i and v_j as multiple edges in the graph, as discussed earlier, we can reconstruct the set of classes in which v_i and v_j have coenrolled when necessary by calculating the intersection their respective sets of courses.

Figure 1: Flow of data and development structure of this thesis.



Now, let us take a step back and examine the structure of the network we have described. Let us call the network $G_s = (V_s, E_s)$, where the set of vertices V_s consists of students, and edges E_s exist between students that have taken a class together. Technically, a student is coenrolled with themself for every class they take, and the weight of this loop is well-defined by 1. However, it's unclear what the practical significance of these loops are for analysis, so we exclude them. Therefore, G_s is a simple, undirected graph.

We also wish to explore differences between various segments of students. Formally, we may define a segment as (S, R), where S is a finite set into which students are mapped, and R is a relation where $R \subseteq V_s \times S$. For example, we may define a "college" segment, where S_c is the set containing all schools, and $(v, \text{school}(v)) \in R_c$. In this case, R_c is a function, every individual maps exactly to one college (yes, even Computer Science majors in LSA). However, R_c need not always be a function. For example, in a "majors" segment, a student could be mapped to several majors. That said, R_c must always be left-total; that is, every student must be mapped to a value. Practically, most of the segments we'll study are defined by functions.

We may also consider the graph $G_c = (V_c, E_c)$, where V_c is the set of courses. G_c is sort of the flip-side of the bipartite network. Instead of projecting the bipartite network onto students as we did with G_s, we project it onto courses. G_c is a simple, weighted, undirected graph, where each weight of an edge, $w(v_i, v_j)$, is an integer equal to the number of students that enrolled in both v_i and v_j. While we do discuss building G_c later, most of our analysis focuses on G_s. Our experiments with G_c do not factor significantly into the results we present in this thesis.

2 Implementation

2.1 Overall implementation structure

To understand the overall implementation structure, it is useful to think of this thesis in the stages as given by Figure 1.

Roughly, the code in this thesis translates us from one stage to another, so code is required at 1 and 2. This code consists of:

1.
   - Building networks.
   - Storing networks.
2.
   - Network processing to output relevant data.

- Visualization creation.
- Statistical analysis.

We'll review the implementation of this thesis using these categorizations.

2.2 Making and storing networks

The heavy-lifting that is needed to create and manipulate the networks is written in C++. C++ was chosen for both my familiarity with it, its speed, and the availability of the Boost Graph library, part of the well-regarded Boost family of libraries [7], which ships with several built-in network analysis algorithms and provides both adjacency matrix and adjacency list implementations for stroing graphs.

Beyond the Boost Graph library, I found a few other Boost libraries useful. The Boost Optional library provides a useful data structure for interfacing with the networks and the Program Options library allows easy parsing of command line arguments and options. Finally, Boost's Serialization library allowed the networks to be stored and read easily and quickly.

Boost itself also satisfies a critical criterion for use: it is available on Flux, the University of Michigan's Advanced Research Computing (ARC) high performance computing cluster. The University of Michigan provides Advanced Research Computing services for researchers who wish to run data-intensive and computational research that cannot be done locally. Flux is one service provided by ARC: it is a computing cluster whose resources researchers may tap into to run computing jobs. Individual instances are available for researchers to purchase; these instances include machines with large amounts of memory or GPU processing capabilities. Public allocations are also available for use when processing needs are moderate. Because G_s is too large to fit in locally in memory, I needed Flux to load the network and output relevant data. While Flux proved essential to this project, it placed a restriction on my development process: all tools for C++ development needed to be available on Flux. With this in mind, we quickly tour the remaining C++ tools I used.

g++ (the C++ compiler in gcc, the GNU compiler collection) compiled and linked C++ code for this project. For local development, the latest stable version of g++, 4.9.2, was used. Unfortunately, the latest available version on Flux was g++ 4.8.0. This older version disallowed the use of newer C++14 features that the gcc 4.9 series begins to implement.

CMake, the cross-platform, open-source build system, managed C++ binaries. While not essential to the development process, CMake allows easier management of code than more traditional tools like Makefiles. The Flux environment provides several versions of CMake.

gtest, Google's C++ unit testing framework, was used to write unit tests for this project. While not as essential as they are in production code, I, along with the rest of the software engineering community, consider them essential to be able to make the claim that code is correct. Additionally, they make targeting errors that are inevitable in the development process easier and assure that old code does not break when new functionality is added.

Because I only used gtest as part of the development process, it is not strictly required in Flux. However, gtest is easy to download and store in a directory for an individual user, which allowed its use in the Flux environment.

2.2.1 Storing networks

To store networks, I created a `Network` class, which wraps a Boost Graph library object that is responsible for storing the actual network structures used in this project.

The `Network` class is a class templated on `Vertex` (the vertex type for the network) and `Edge` (the edge type for the network). It contains only one data member, a Boost adjacency matrix that holds the actual data in the network. There are several design reasons for implementing a wrapper class.

1. **Easier, more idiomatic access to vertex and edge values:** The Boost Graph library provides an outdated mode of iterating over vertices and edges: macros. A carryover from the C era, macros are nearly universally frowned upon in modern C++ [27]. As opposed to using these dinosaurs, I took it upon myself to allow a wrapper class to access vertex and edge values in a more idiomatic way, using "pseudo"-containers and a new iterator.
2. **Providing an appropriate level of generality:** As mentioned earlier, the Boost Graph library includes implementations of adjacency list and adjacency matrix. The adjacency list representation takes seven template parameters, the adjacency matrix five. While this ability to customize makes the Boost Graph library useful in many different systems, it is overkill for this thesis. The `Network` class uses the adjacency matrix representation due to the fact that both the course and student network are dense graphs. It provides two template parameters, as mentioned previously.
3. **Free functions vs. member functions:** The Boost Graph library relies mostly on free functions, as opposed to member functions, to perform actions on graphs. For example, to get an edge between two vertices in a graph, one uses the function `boost::edge(u, v, g)`, where `u` and `v` are vertex descriptors and `g` is a Boost graph object. It is worth mentioning that the C++ community seems to be adopting free functions over member functions recently, e.g. the free versions of `begin` and `end` are preferred over the member versions for more general code [28]. In fact, a paper has even been submitted that suggests C++17, the next version of the C++ standard, support member functions called with a specific free function syntax [29]. Regardless of this debate, the wrapper class must provide access to the inner Boost graph object, or provide wrapper functions for it. Good programming style encourages classes to be thought of as abstractions; they should not be tied to specific implementation details. For this reason, permitting access to internal members is generally considered bad style. I decided to reimplement the graph functions as member functions of `Network` for the increased clarity they offer as to which object is having actions performed upon it.

Modernizing Boost Graph library code: As mentioned in item 1 above, updating the Boost Graph library code to use more modern C++ as opposed to macros was a motivating factor in deciding to wrap the Boost Graph object. The design of this feature was not trivial for several reasons.

1. **The solution should allow the use of the standard library:** While the Boost Graph library does provide iterators to iterate over indices (called "descriptors") of vertices and edges, we also want to be able to iterate over the values. Iterating over the values by means of the descriptors results in clunky code. For example, if we wished to copy the values from the graph object and had the ability to iterate over values, we could use a semantically clear call to `copy()`. Iterating over descriptors means that our call to copy must be changed into a call to `transform()`, which is not quite as self-documenting. Therefore, the solution should allow us to use a graph's vertex and edge values as I would any standard library container.

2. **The solution should be general:** Less code introduces fewer bugs. Therefore, instead of implementing two different solutions: one to access vertices and one to access edges, I wanted a single solution. A single solution also adapts to other similar issues. For example, I realized later that I wanted to be able to loop over the adjacent edges of a vertex. The solution should require a minimal amount of code to adapt to this new situation.

3. **The solution should be `const` correct:** That is, the solution should allow the user to, for example, change the values stored by vertices only when given a graph that is mutable, but enforce read-only access when the graph is `const`.

A solution that solved these three criteria involved implementation of several nested classes in `Network`. A UML class diagram is given in Figure 2.

Interface of the solution: As this figure demonstrates, the `Network` wrapper implementation is nontrivial. It is most useful to consider the classes `Descriptors` and `Values`, which serve as the only interfaces used by consumers of the `Network` class.

`Descriptors` is a class that allows iteration over descriptors, for example, vertex or edge descriptors, in the graph. As mentioned previously, a descriptor of a vertex is an index that the Boost Graph library uses to uniquely identify a particular vertex.

I also wanted the ability to also iterate through the values contained by graph vertices/edges, so I encapsulated this functionality in the `Values` class.

The base class of these two derived classes is the `Properties` class, which simply provides the `size()` function that should be given by the `Adaptor` object. I decided that `size()` should be a part of the interface because the `Descriptors` and `Values` classes act as pseudo-containers, and most containers in the C++ standard library implement the `size()` function. In fact, a proposal has been submitted to the standards committee suggesting that a free `size()` function be added to the standard library for the next standard of C++ [21].

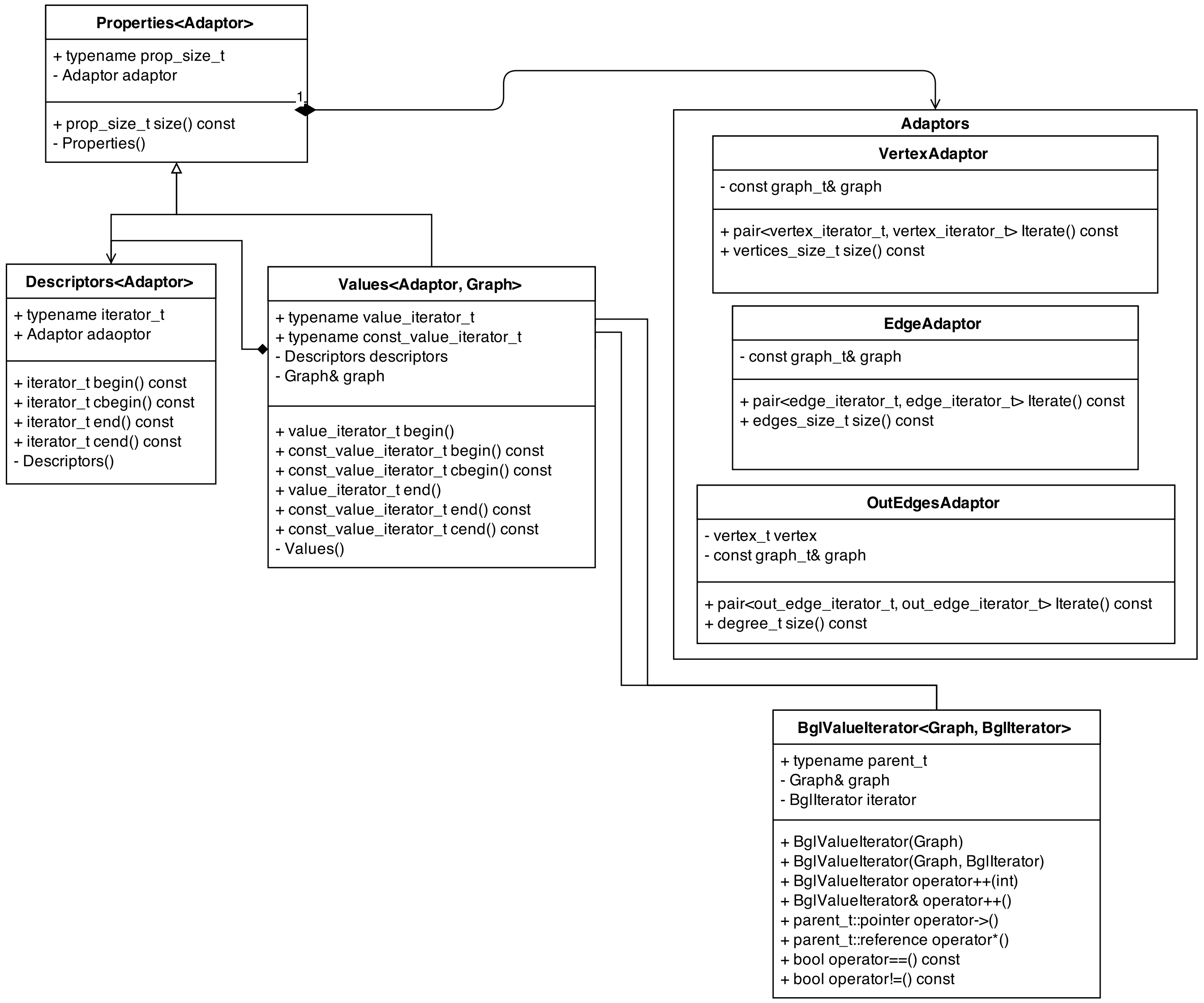

Properties<Adaptor>
+ typename prop_size_t
- Adaptor adaptor
+ prop_size_t size() const
- Properties()
1
Descriptors<Adaptor>
+ typename iterator_t
+ Adaptor adaoptor
+ iterator_t begin() const
+ iterator_t cbegin() const
+ iterator_t end() const
+ iterator_t cend() const
- Descriptors()
Values<Adaptor, Graph>
+ typename value_iterator_t
+ typename const_value_iterator_t
- Descriptors descriptors
- Graph& graph
+ value_iterator_t begin()
+ const_value_iterator_t begin() const
+ const_value_iterator_t cbegin() const
+ value_iterator_t end()
+ const_value_iterator_t end() const
+ const_value_iterator_t cend() const
- Values()
Adaptors
VertexAdaptor
- const graph_t& graph
+ pair<vertex_iterator_t, vertex_iterator_t> Iterate() const
+ vertices_size_t size() const
EdgeAdaptor
- const graph_t& graph
+ pair<edge_iterator_t, edge_iterator_t> Iterate() const
+ edges_size_t size() const
OutEdgesAdaptor
- vertex_t vertex
- const graph_t& graph
+ pair<out_edge_iterator_t, out_edge_iterator_t> Iterate() const
+ degree_t size() const
BglValueIterator<Graph, BglIterator>
+ typename parent_t
- Graph& graph
- BglIterator iterator
+ BglValueIterator(Graph)
+ BglValueIterator(Graph, BglIterator)
+ BglValueIterator operator++(int)
+ BglValueIterator& operator++()
+ parent_t::pointer operator->()
+ parent_t::reference operator*()
+ bool operator==() const
+ bool operator!=() const

Figure 2: A UML diagram of various nested classes in `Network` and their dependencies.

The two derived classes also provide `begin()` and `end()` member functions, which allow them to be used with standard library algorithms.

Implementation of the solution: The classes `Values`, `Descriptors`, and `Properties` make use of adaptors to make the Boost Graph library interface align with the interface they present. Specifically, an adaptor to any one of these classes must provide the functions:

- `Iterate()`: Returning a pair of iterators, the first pointing to the beginning of the set of elements, the second pointing to the end.
- `size()`: Returning the size of the set of elements.

These two functions were chosen as an interface for the adaptors because they are similar to what the Boost Graph library has as an interface. For example, `VertexAdaptor`, which allows the vertices of the graph to become a pseudo-container, uses the Boost Graph library's built-in `boost::vertices()` function to supply the result of `Iterate()`. `boost::num_vertices()` supplies the result of `size()`.

The use of adaptors as template parameters to the `Descriptors` and `Values` class allows easy extension of these classes to allow iteration across a variety of different objects. As I mentioned earlier, originally, I wanted to be able to iterate across vertices and edges. However, upon further development of this thesis, I realized it would be useful to iterate over edges adjacent to a specific vertex. Adding this functionality was as simple as adding `OutEdgesAdaptor`.

The adaptor pattern that I have described is usually a polymorphic hierarchy, where every individual adaptor provides an implementation of functions declared pure virtual by the base adaptor class from which it inherits. In this project, the classes must be template types instead because the function signatures are different in each class. That is, while both `VertexAdaptor` and `EdgeAdaptor` provide an `Iterate()` function, the first returns a pair of vertex iterators, while the second returns a pair of edge iterators.

It's also worth discussing the use of the template parameter `Graph` in the `Values` class. While we know the type of `Graph`, we do not know whether this type will be `const` or non-`const`. We want the values over which we iterate to be mutable while the `Network` class is non-`const`, but to be immutable when `Network` is `const`. While it seems somewhat like overkill to use templates simply to achieve `const`-correctness, I could not conceive of another solution that would be more elegant. I considered storing the pseudo-containers of vertices, edges, etc. as members of the class, which would guarantee their `const`-correctness, but clutters up the class with members that must be initialized in the constructor. Also, the inelegance of this solution becomes clear when considering the `OutEdges` pseudo-container. An `OutEdges` pseudo-container member would have to exist for every vertex in the class, because a consumer of `Network` should be able to construct an `OutEdges` pseudo-container for every vertex. To store all these pseudo-containers would require an unreasonable amount of space. Therefore, I chose to accept `Graph` as a template parameter, which allows creation of the pseudo-containers as requested by consumers of `Network`.

The UML diagram in Figure 2 also shows that `Values` includes a `Descriptors` object. The Boost Graph library allows one to access the values stored by vertices and edges by calling `graph[descriptor]`, where `graph` is the Boost Graph library object, and `descriptor` is a vertex or edge descriptor. The function returns the corresponding vertex or edge value. Because a descriptor is required by the Boost Graph library to access the corresponding value in the graph, `Values` stores a `Descriptors` object and uses it to access the underlying values stored in the graph object.

Assisting in accessing the values stored in the graph object is `BglValueIterator`, which derives from the `iterator` class in the C++ standard library. This iterator class stores the underlying descriptor and also keeps a reference to the graph object, and actually calls the `graph[descriptor]` function to return the descriptor's corresponding value. Ultimately, the `Values` class just creates a `BglValueIterator` and delegates responsibility of iteration to it.

2.2.2 Building networks

Network building was also a nontrivial task. G_c, a dense graph of ≈ 8000 vertices, is small enough to build locally. G_s, on the other hand, a dense network of ≈ 58000 vertices, required additional computing power. A quick back of the envelope calculation using $m = n^2$ edges gives the space estimate:

$$58000^2 \text{ edges} \times 4 \text{ bytes/edge} \approx 13 \text{ gigabytes}$$

to store the values in the edges. In reality, the graph structure requires more memory to store the values in the vertices (though this is orders of magnitude less memory than to store the edge values) and also to store the descriptors for the vertices and edges. Ultimately, when completed, the full graph requires $\approx$ 25GB of memory, certainly not an unreasonable amount, but far greater than the 8GB in my notebook.

At this scale, choosing an algorithmically efficient way to build the graph also becomes incredibly important. This will become clear as I recount the struggle to build this network.

Building G_c: Due to the relatively small size of G_c, it was relatively easy to build in a reasonable amount of time without needing to make smart or clever choices regarding algorithms. Basically, the approach was:

1. Create a map of student ID to the list of courses that the student took.
2. For every pair of courses that a student has taken, add +1 to the weight of the edge between those two courses.
3. Create the graph structure: courses are vertices, and assign the calculated weight of the edges.

On my local machine, this process took ≈ 5 minutes. Fortunately, the Boost Graph library includes serialization functionality, so the network does not have to be rebuilt every time it is loaded into memory. Recreating the course network from the serialized data took ≈ 5 seconds.

Building G_s

A first pass: The most naive approach to building G_s might take an analogous approach to that used for building G_c. Specifically:

1. Create a map of course to students enrolled in that course.
2. Increment their edge weight by an appropriate amount for every pair of students in every course.
3. Create the graph structure with students as vertices and edges as calculated.

Indeed, the first attempt I made at building this graph used the process described by the above steps. After the code ran for about 8 hours on Flux I decided that I needed to better understand how close the code was to completion. It was at this point that I began to run benchmarking.

I found several issues with the method above. Most immediately salient among these is the difficulty in determining the scaling of this method. Because classes have varying enrollment sizes, the time to process every class is not constant, and therefore is not a good unit for understanding the scaling of this method overall. To remedy this issue, I decided to count pairs of students processed as the basic unit of scaling.

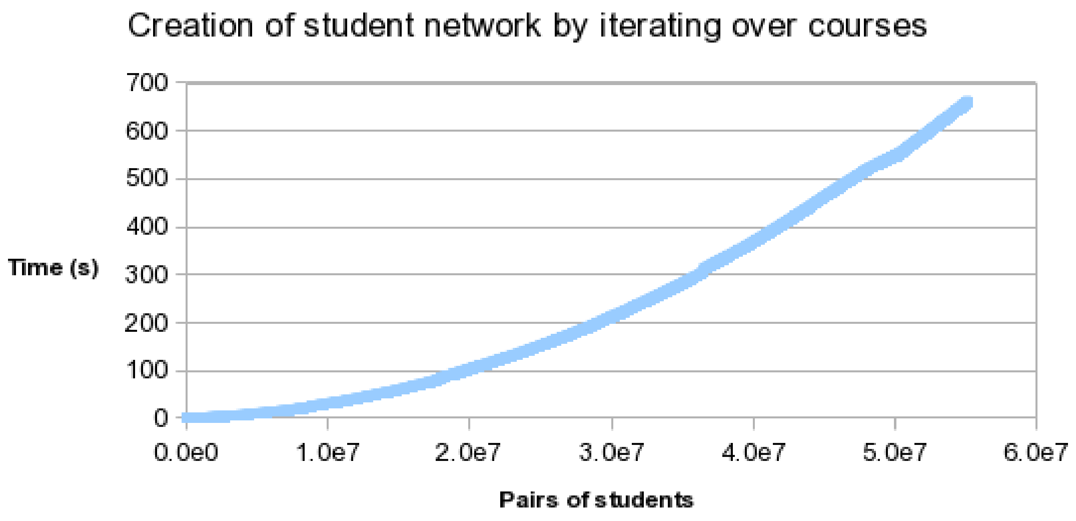
This led me to discover a surprising issue: the scaling was clearly quadratic. The the time to process a pair increased proportional to the number of pairs squared, as opposed to the number of pairs, as I expected. This is clearly shown in Figure 3a, which shows what looks like a quadratic relationship, and Figure 3b, which shows a linear relationship between the square of pairs of students and time. To this day, this issue baffles me. The only step that I took when processing the pairs was to insert them into a hash table indicating that they were an edge, an operation that should take $O(1)$ time, regardless of the size of the hash table. While it's true that the performance of hash tables does depend on the hashing function, quadratic scaling signifies remarkably bad performance. Had I more time, I would like to further investigate this issue.

I ultimately discarded this method of building G_s for another reason. I hypothesized that by looping over classes and adding students, I was likely hitting the same edges multiple times, and thus I incurred extra lookup cost for every duplicate pair of students.

My benchmarking data also suggested that this first attempt at construction would not be fruitful. When I extrapolated my sample to all pairs of students that I would have to process, on the order of about a billion, I calculated time estimates on the order of hundreds of hours.

Figure 3: Scaling performance of initial attempt to build G_s. This shows a small sample of pairs of students. Overall, there are on the order of a billion.

(a) A graph showing the scaling performance of iterating over classes to build G_s.



(b) Figure 3a normalized to show $O(n^2)$ time complexity.

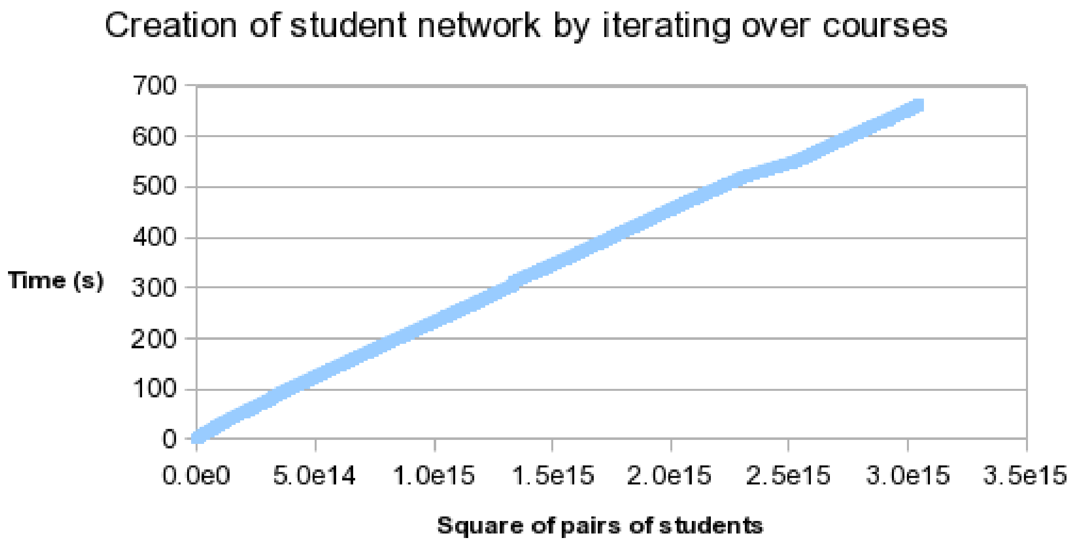
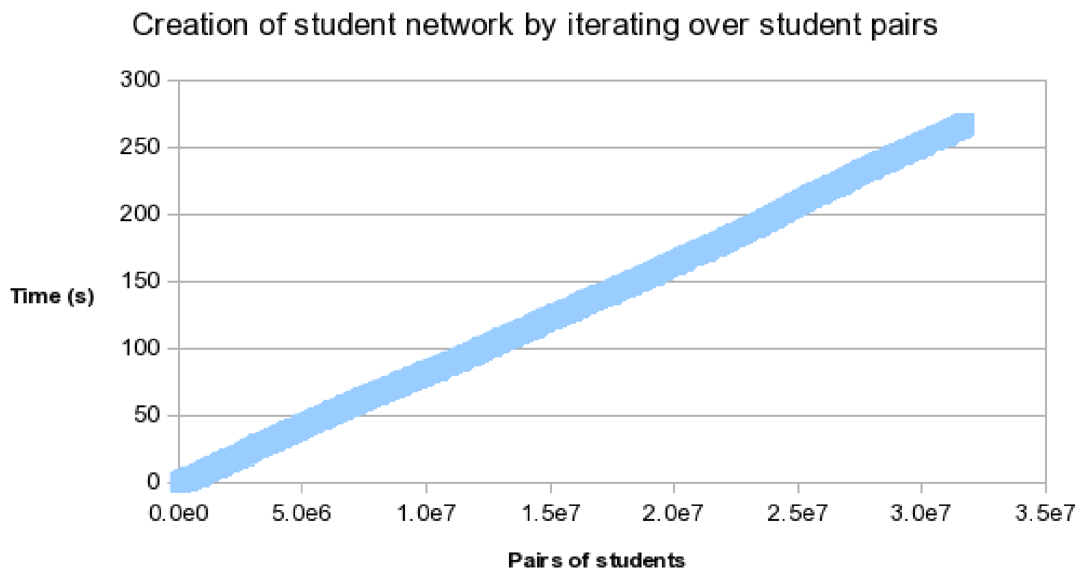


Figure 4: A graph showing the scaling performance of iterating over pairs of students to build G_s.



Switching construction methods: My conception for a new method took the following steps:

1. Examine every pair of students.
2. Calculate the value of the edge for the pair of students by looking at the intersection of their course schedules.
3. Create the graph structure with students as vertices and the edges as calculated.

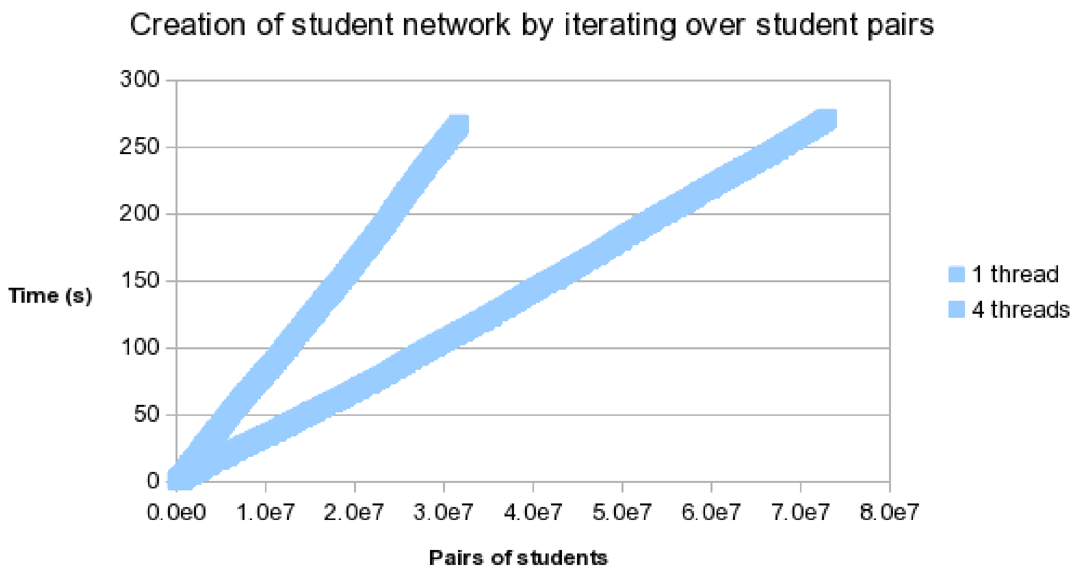
This method has several properties I suspected would be advantageous:

- **It determines the value of edges immediately.** While the other method adds on to edge values as the same pair of students is discovered in multiple courses, this method calculates the value of the edge between students immediately and makes no further modifications. This means that any lookups to get the previous value of the edge are eliminated.
- **It avoids the troublesome hash table.** Without the requirement that edges must be looked up, the hash table that was used in the previous method may be replaced with a more reliable data structure, like a simple array.

Indeed, upon switching to this method, benchmarks were much more promising. Figure 4 demonstrates a linear scaling time, and extrapolating to all pairs of students gave time estimates of approximately 1.5 hours for calculating all the edge values.

To speed this up even more, I added some basic parallelization to the edge value calculation. This yielded expected multiplicative performance improvements, see Figure 5.

Figure 5: A graph showing the scaling performance of iterating over pairs of students to build G_s with different numbers of threads.



The importance of data structures: One additional challenge remained: The edges actually needed to be inserted into the graph structure. It was here that the performance differences between adjacency lists and adjacency matrices became very visible. I had been developing with the adjacency list due to the fact that more Boost Graph library examples were provided for it. The switch to adjacency matrix was relatively easy, and yielded remarkable performance improvements, see Figure 6. This is to be expected, changing from a data structure where the cost of inserting an edge is $O(m/n)$ to one in which insertion takes $O(1)$ time obviously shortens the overall runtime quite a bit. In fact, benchmarks showed that by using the adjacency matrix representation, I cut the time to build the graph down from over eight hours to a few minutes.

Building the network in-place: Building the graph in place allowed me to realize another performance gain when building G_s. My original method of building the network by iteration through courses precluded this possibility by requiring easy lookup to modify edge values. My newer method of iterating over all pairs of students to calculate edge values in one pass allowed me to insert them into the network immediately, no intermediate data structure required. This cut overall time to build the network to about 15 minutes, as demonstrated by Figure 7.

Difficulties with Boost's implementation of the adjacency matrix: While the switch to adjacency matrix as mentioned earlier was relatively easy, I did notice a few peculiarities. It appears as though much more development has been done for adjacency lists. The adjacency matrix graph implementation lacks several features its list cousin has.

Figure 6: A graph showing the scaling performance differences between inserting edges into an adjacency list vs. an adjacency matrix.

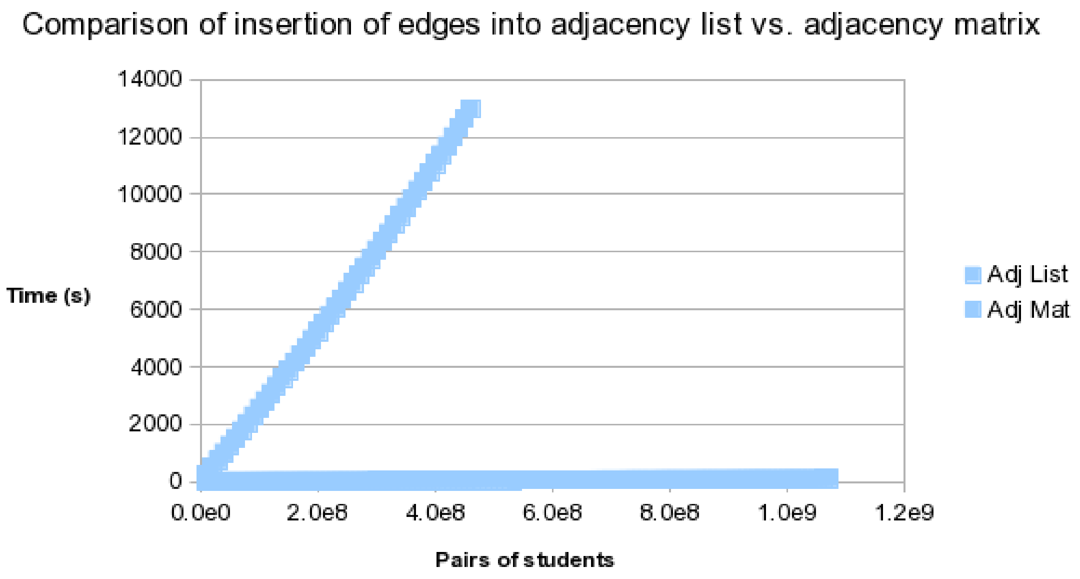
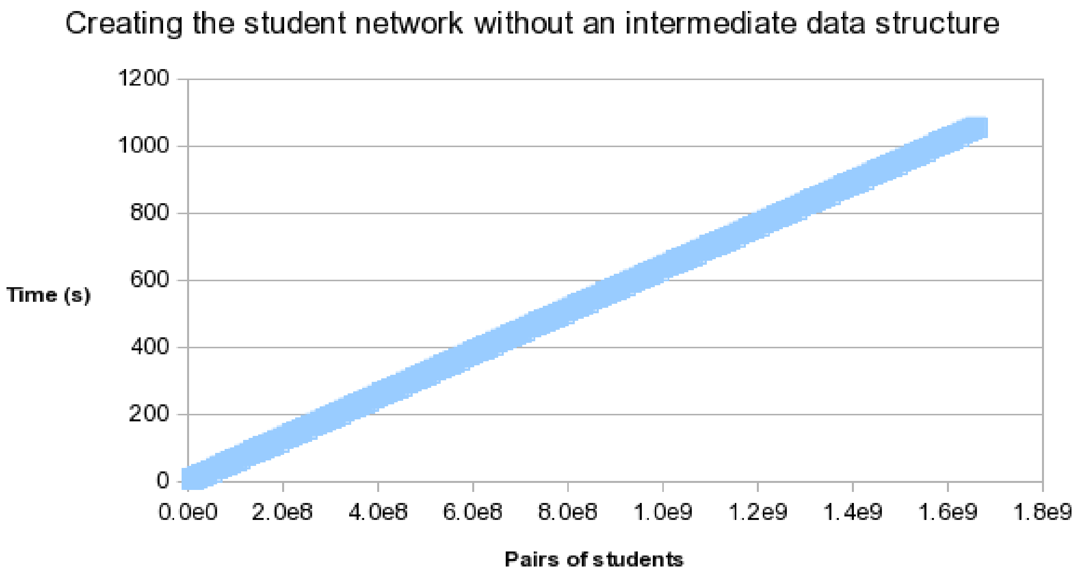


Figure 7: A graph showing the scaling performance building the student graph in place without an intermediate data structure.



Perhaps most obvious is that graphs constructed with adjacency matrices do not have the ability to add vertices. This means that when a consumer of the Boost Graph library creates a network, they either must know the number of vertices it will have, or incur the large cost of replacing the original graph with a new graph when they do know how many vertices they want. Ultimately, this is a minor inconvenience, especially for my case, where the number of vertices is obviously equal to the number of students.

A larger issue with graphs implemented with adjacency matrices is that they lack built-in serialization functionality. The ability to serialize the network to a file so as not to build the graph every time the program is run is essential. I had to create my own serialization functionality for adjacency matrices, which I heavily based off the existing serialization functions for adjacency lists. In fact, I only needed to change a few function calls to work around functionality that Boost's adjacency matrix functionality lacked.

Finally, on the version of Boost that Flux has, the function to add an edge when its value is not given is incorrectly implemented. It does not initialize the edge structure, it declares (`EP ep`) instead of initializing (`EP ep{}`) leading to compiler warnings when the edge structure has a fundamental type. Fortunately, the workaround for this was simple: provide an edge value to another overload of the function that is correctly implemented.

These issues present with Boost's implementation of adjacency matrix suggest that Boost's adjacency matrix implementation for graphs has received less development attention than adjacency list has. While the development priorities of Boost are an aside to the focus of this thesis, most of these problems seem relatively easily fixed, and this observation offers interesting insight into the priorities of the developers of a large and popular library.

2.3 Analysis code

C++ isn't particularly well-known for data analysis. More often, statisticians and data analysts opt for languages like R. Having little experience in R, I instead chose Python 3 for most of my analysis tasks. While Python 2 still remains very popular for compatibility reasons, I chose Python 3 because the libraries I planned on using had been upgraded to support it, and I didn't rely on any legacy code.

I used the following Python libraries. Because I ran my analysis code locally, Flux constraints were not a concern for me.

- **NumPy:** A popular Python library known for fast numerical operations on n-dimensional arrays.
- **matplotlib:** A Python library that allows the easy creation of graphs and other visual representations of data.
- **SciPy:** While most of my analysis was not incredibly heavy on statistics, I used SciPy because it bundles in a lot of statistical analysis functionality.

2.3.1 Network processing to output relevant data

From a design perspective, this was by far the most tricky part of the analysis portion of my work. Determining how to organize, integrate, and split responsibilities between C++ and Python code posed many design challenges.

Specifically, because all the network heavy lifting was handled by the C++ code and could not be run locally, I had to anticipate which data I would need from the network, and output it into a format into which I could easily analyze with the Python code.

For example, many parts of the analysis consist of segmentation of individuals (refer to Section 1.7 for a formal definition). When should the segmentation of individuals happen? Should it happen on Flux, when the C++ binary outputs information we need for analysis? Should it happen locally, when analysis is done?

Originally, I chose to do it on the Flux side, reasoning that because I had all the student information stored in memory anyway, I could anticipate the segments we would want to analyze in advance and output several copies of the data segmented in different ways.

This eventually revealed itself as less than ideal. Short term issues included cluttering up my C++ binaries with filenames for all the output. To a certain extent, this was inevitable: it's difficult to write general code that outputs specific data. However, I realized that this was not a solution when I began exploring connections of individual students. Creating separate files for various segmentation schemes when exploring the connections of just 100 individuals would balloon the number of files to several times that. Most of this data just consisted of numbers copied from file to file with a different label in front of them.

Choosing to segment files on the fly presented other issues. Specifically, doing any sorts of graphic or visualization work in C++ is a difficult and pointless exercise, especially when readily available and easily-used Python libraries exist. On the other hand, all my data on students and classes was read in by my C++ binary, inaccessible to my Python code. Either I could rewrite the functionality to read the input data, essentially duplicating uninteresting, mind-numbing work in another language, or I could use a C++ binary as an intermediary between the data output by Flux and my Python analysis code.

Using a C++ binary as an intermediary presents its own issue. When the binary is run for every output file, it must reread the input data over and over again. While this takes just a few seconds, when scaled to run over several hundred files, it is a bottleneck in the analysis process, where we the development cycle is fast, because we wish to make slight tweaks and see how the results differ.

After descending into this rabbit hole of problems and half-solutions, it is useful to take a step back and consider how this problem could have been solved from the start. My distillation of the problem is: how can certain applications get bits of information they need when it is stored in a central repository? This problem has actually been mostly solved by the programming community: use an application programming interface (API). In the age of mobile applications and webapps, these are mostly hosted over the Internet. Scaling this down would have been a perfect solution for my issue. I could have hosted a simple C++

server that provided information on students and courses to clients upon request. Boost provides the ASIO library for this, so it certainly feasible. Failing to see this solution and implement it earlier is my biggest design flaw in my thesis code, and is certainly something I would do differently if presented the opportunity to start over.

2.3.2 Visualization and statistical analysis

Once data has been properly processed for analysis, creating visualizations is just a matter of understanding the libraries well enough. No interesting code design problems arose here, so I'll only provide a brief description of what I used to create visualizations.

Most of the visualization and analysis took place using Python 3. Notably, the availability of the popular `matplotlib.pyplot` library for creating graphs made Python a clear choice for parsing and creating the figures included later in this report. I also originally explored Python as a choice for building the networks due to its ease of use, but eventually abandoned it in favor of C++.

`matplotlib.pyplot` provides `histogram` and `pcolor` functionalities, which I used to create the histograms and heatmaps that will appear in Section 3.

The chord diagrams are an interesting and useful visualization `matplotlib.pyplot` does not create. These diagrams visualize a matrix of values by arranging row and column labels on a circle and drawing chords between them with a thickness that represents the value in the corresponding cell. To create these, I used d3, a well-known Javascript library for data visualization. Creating diagrams with d3 also allows the diagrams to interact with mouse movements. Unfortunately, my lack of familiarity with d3 prevented me from utilizing it to a greater extent for data visualization. This should certainly be a focus of future work; the d3 community has created a myriad of compelling visualizations that could be used to better understand the networks.

3 Analysis

In this section, we'll pull various views of data from the entire student network G_s. One way we examine student experience is by looking at various groups of students and examining how they are connected using their degrees (Section 3.2). The degree of a vertex in an undirected graph is the number of edges connected to it [23]. This definition is inherently unweighted, it considers all edges equivalent. Because considering the weighting of edges is important to us, we will call this traditional definition of degree the "unweighted degree". We will also use a "weighted degree" measure, which we will formally define later. Examining distributions of weighted and unweighted degree for different segments of students can reveal some patterns about certain groups of students, like whether a student's number of connections to other students varies by program. While we do present some qualitative theories about some of the results we see in this section when we can, to analyze all of them requires more of a sociological analysis, which lies beyond our capabilities.

In Section 3.3, we perform an analysis of the interactions between segments of students. We rely on a comparison with a random graph to determine whether interactions occur more or less often than we would expect. Random graphs are networks where some properties are fixed, but some properties are allowed to vary randomly [23]. Comparison to random graphs is a useful way to analyze particular properties of networks [23]. In practice, as we'll see in Section 3.3, we do not need to actually build the random graph, we just need to calculate expected values on some of its properties. Like Section 3.2, we'll see that in this section, we are not always able to justify the results we calculate without a deeper sociological analysis.

However, the analysis done in Section 3.3 points us toward a metric we can use to achieve our original goal: measuring a student's intellectual connectedness. We do this explicitly in Section 3.4. We'll present a metric that we believe captures diversity of student connections, and we'll look at individual students and try to reconcile the value the metric gives to them with their course schedule. We'll look at examples of highly connected, typically connected, and less connected students from various groups.

3.1 Basic properties of G_s

When discussing G_s, it is often informative to look at two different versions of the measures from the network, which we will call "weighted" and "unweighted". The "weighted" network is G_s as we have described it previously: with weights by students given by (1). The "unweighted" network is G_s with all edges between students normalized to a weight of 1. This network provides information on just the number of connections a given student has. This network has a bias toward students who take courses with larger enrollments, but we study it to get an idea of how many people an individual student is connected to based on the courses they take.

As mentioned before, G_s is a dense graph. Precisely, it has $57,752$ vertices (the same as the number of students given), and $127,163,500$ edges.

One easy property of G_s to study is the degree distribution. We present two degree distributions, an unweighted and a weighted distribution in Figure 8. Let us define "weighted degree" of a vertex v as $\deg_w(v) : V \to \mathbb{R}$, whose definition is given in (2).

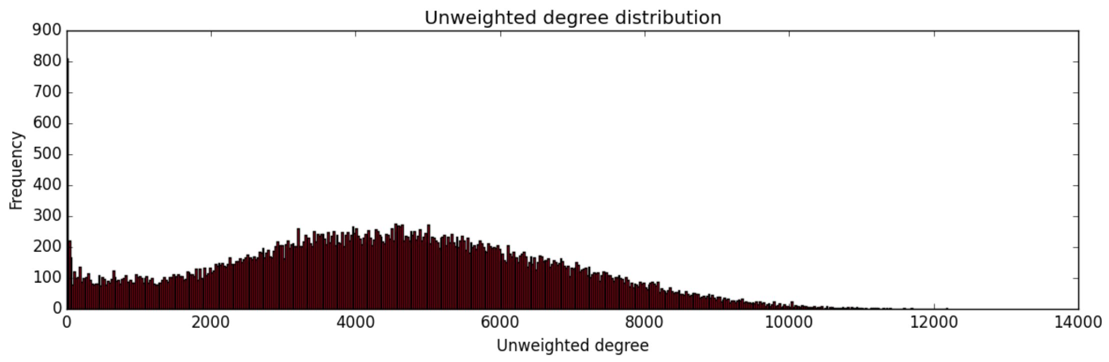
$$\deg_w(v) = \sum_{u \in \mathrm{adj}(v)} w(v,u) \tag{2}$$

Let $\mathrm{adj}(v)$ be the set of vertices adjacent to v, and w the weighting function of G_s. Conceptually, the weighted degree of a vertex v is the summation of the weights of the edges adjacent to v.

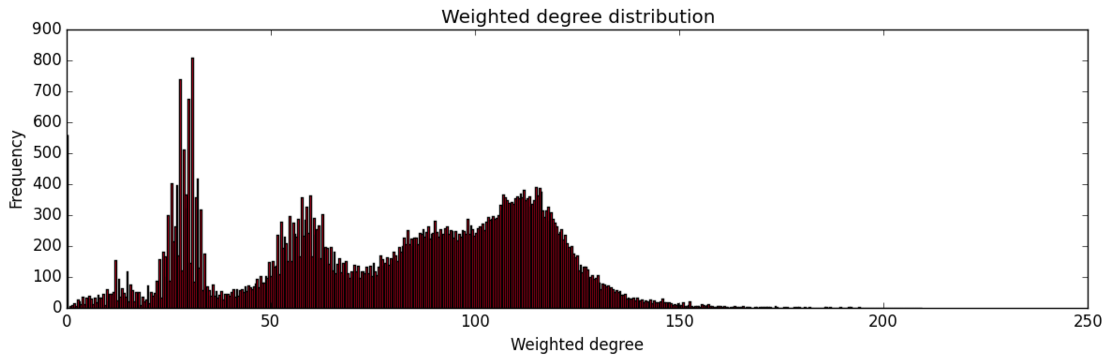
In the histogram in Figure 8a, we see that the distribution appears to be roughly symmetric, with a shallow peak occurring in the center and long sloping tails. The histogram in Figure 8b tells another story. We see two distinct peaks which appear to occur around 30, 55. Two less pronounced peaks occur at 90 and 115. Both histograms appear to have a large

Figure 8: Degree distributions of G_s

(a) A histogram showing the unweighted degree distribution of G_s.



(b) A histogram showing the weighted degree distribution of G_s.



population at the 0 mark as well. We will further analyze these distributions in Section 3.2. In particular, Section 3.2.1 will elucidate the weighted degree distribution both theoretically and emprically.

In his book, Newman discusses many other measures and metrics for networks [23]. Unfortunately, G_s is a rather large network, and many of these measures are infeasible to compute in a reasonable amount of time.

3.2 Degree summations for various segments of students

To help unpack the data in Figure 8, we can consider different segments of students and look at their degree distributions.

3.2.1 Students segmented by first year

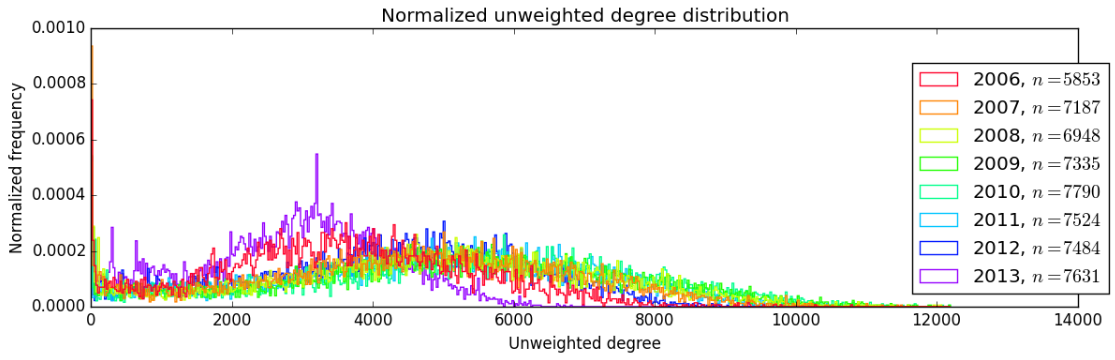
First, we consider students segmented by the first year they enrolled. Studying these segments yields rather illuminating results on the shape of the overall distribution. In Figure 9 we show normalized histograms. In normalized histograms, the area under the distribution sums to 1. Normalization allows us to make comparisons between the distributions without the size of the segment distracting us. While not entirely necessary when studying this segmentation pattern because the segments are all similar sizes, it is indispensible when comparing distributions for segments with very different sizes. In this section, we will generally only show normalized histograms when comparing segments.

Figure 9b is particularly interesting. We see several distinct peaks at around 30, at 60, 90, and 115. When we rewrite the summation of the weights of all individual connections for each student, the reason becomes clear. Recall that E_c is the set of students enrolled in class c, and H_c is the number of credits class c offers. As we have defined before, let $\deg_w(v)$ be v's weighted degree. Let C_v be the set of courses student v has taken. Let S_v be the set of students coenrolled with v, that is:

$$S_v = \bigcup_{c \in C_v} \{u : u \in E_c, u \neq v\}$$

Figure 9: Degree distributions of students by first year.

(a) A histogram showing the unweighted degree distribution of students segmented by first year.



(b) A histogram showing the weighted degree distribution of students segmented by first year.

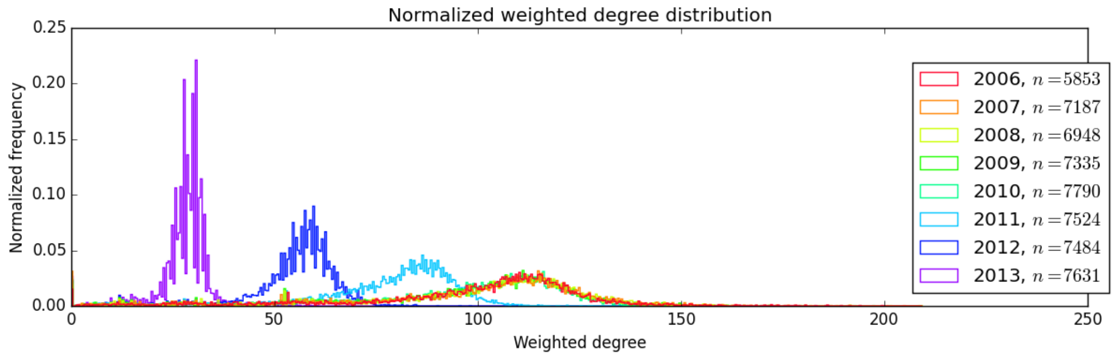
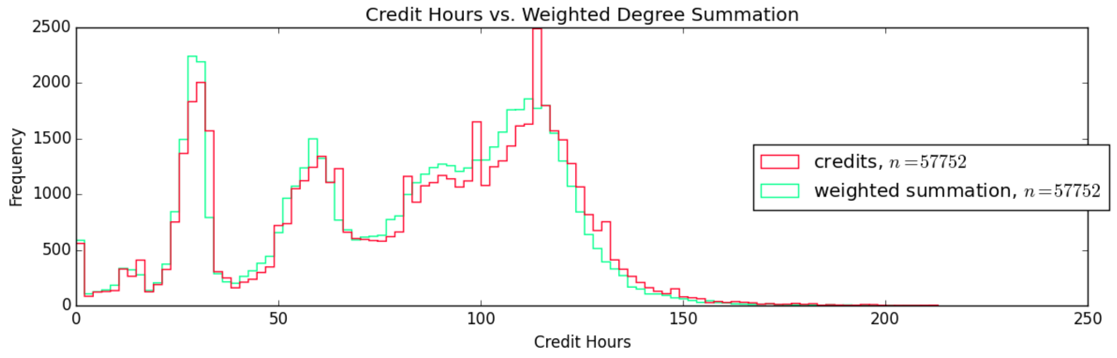


Figure 10: Distribution of weighted degree and credit hours.



Now, we may write $\deg_w(v)$ as:

$$\begin{aligned}
\deg_w(v) &= \sum_{u \in S_v} \sum_{c \in C_u \cap C_v} \frac{H_c}{|E_c|} \\
&= \sum_{c \in C_v} \sum_{u \in E_c - \{v\}} \frac{H_c}{|E_c|} \\
&= \sum_{c \in C_v} (|E_c| - 1) \times \frac{H_c}{|E_c|} \\
&= \sum_{c \in C_v} \frac{(|E_c| - 1)}{|E_c|} \times H_c \\
&= \frac{(|E_c| - 1)}{|E_c|} \times \sum_{c \in C_v} H_c \\
&\approx \sum_{c \in C_v} H_c
\end{aligned}$$

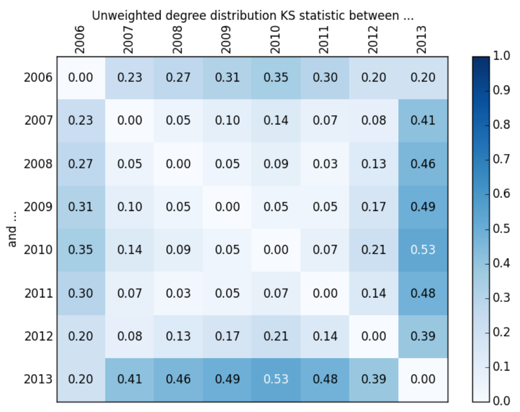
This means that the distribution of degree weights is roughly equivalent to the distribution of credit hours. This allows us to understand the peaks we see in Figure 9b: they are roughly the number of credit hours that a student has completed after one, two, three, . . . years. Once students hit about 120 credits, they begin to graduate, explaining the last peak in the diagram. To further appreciate this relationship, we provide a comparison of the weighted degree distribution vs. credit hour distribution in Figure 10.

Another interesting feature of Figure 9b is that the peaks smooth out as students progress. This makes sense if we consider that students take classes at different rates. The students taking classes at slower rates continue to fall behind, whereas those who take more credits every semester stay ahead.

More difficult to parse is the distribution of connections for unweighted degree segmented by year in Figure 9a. While our eyes are powerful, it would be nice to have a test to see if

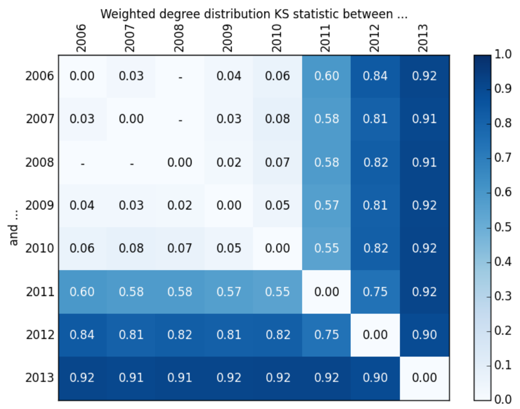
Figure 11: KS statistics for pairs of first years.

(a) A histogram showing the unweighted degree distribution of G_s segmented by first year.



	2006	2007	2008	2009	2010	2011	2012	2013
2006	0.00	0.23	0.27	0.31	0.35	0.30	0.20	0.20
2007	0.23	0.00	0.05	0.10	0.14	0.07	0.08	0.41
2008	0.27	0.05	0.00	0.05	0.09	0.03	0.13	0.46
2009	0.31	0.10	0.05	0.00	0.05	0.05	0.17	0.49
2010	0.35	0.14	0.09	0.05	0.00	0.07	0.21	0.53
2011	0.30	0.07	0.03	0.05	0.07	0.00	0.14	0.48
2012	0.20	0.08	0.13	0.17	0.21	0.14	0.00	0.39
2013	0.20	0.41	0.46	0.49	0.53	0.48	0.39	0.00

(b) A histogram showing the weighted degree distribution of G_s segmented by first year.



	2006	2007	2008	2009	2010	2011	2012	2013
2006	0.00	0.03	-	0.04	0.06	0.60	0.84	0.92
2007	0.03	0.00	-	0.03	0.08	0.58	0.81	0.91
2008	-	-	0.00	0.02	0.07	0.58	0.82	0.91
2009	0.04	0.03	0.02	0.00	0.05	0.57	0.81	0.92
2010	0.06	0.08	0.07	0.05	0.00	0.55	0.82	0.92
2011	0.60	0.58	0.58	0.57	0.55	0.00	0.75	0.92
2012	0.84	0.81	0.82	0.81	0.82	0.75	0.00	0.90
2013	0.92	0.91	0.91	0.92	0.92	0.92	0.90	0.00

the distributions of various segments are measurably different.

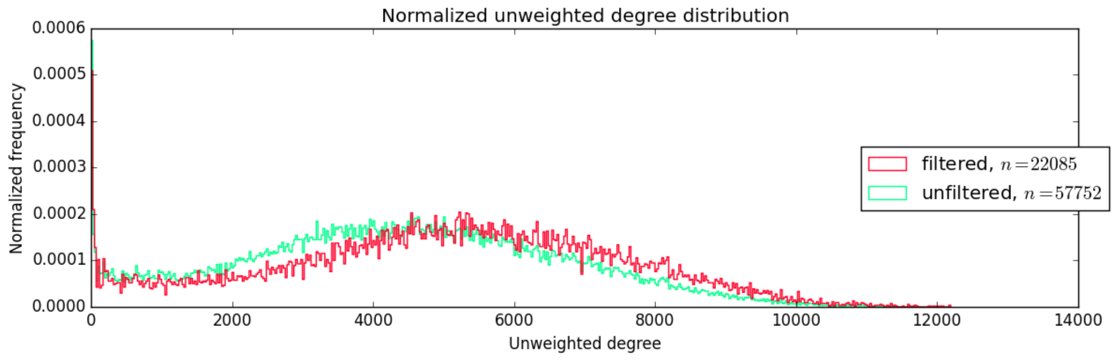
In fact, the Kolmogorov-Smirnov "KS" test can tell us just this. Roughly, the two-sample KS test measures the difference between distributions by computing a statistic that is the supremum of the difference between the distributions at all points. Thus, we use the KS statistic is a proxy for measuring the size of the difference between the distributions. Figure 11 shows a "heatmap" for differences between the distributions of the first years. It also specifies sets a threshold of $p < 0.05$, marking those cells that do not meet this criterion with "-". For completeness, we include KS heatmaps for both unweighted and weighted degree distribution.

A few notes on Figure 11 before we proceed. The matrices it contains are obviously symmetric because the KS-statistic is symmetric. Also not surprising are the 0 entries along the diagonal, meaing that a distribution is the same as itself. These observations serve as sanity checks to ensure that the KS statistic does convey the information we want.

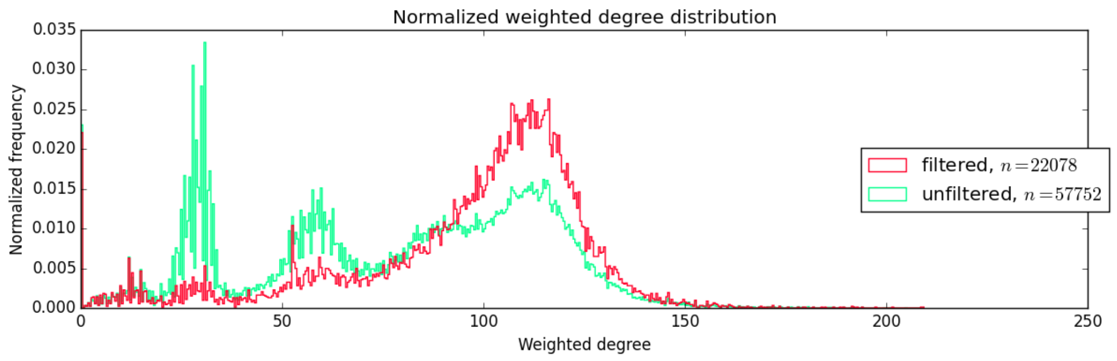
While the KS heatmap in Figure 11b seems to confirm what we observed from the distribution, Figure 11a reveals patterns we may not have spotted in the histogram. This KS heatmap seems to indicate that the degree distribution is similar for those years a similar distance away from the "edges" of the data set: 2006 and 2013. For example, the distributions for 2006 and 2013 are similar, the distributions for 2007 and 2012 are similar, etc. The reason for this is that our data do not include students who entered earlier than 2006, thus missing any connections to them other students may have had. For students who entered in 2006, this means that we are missing connections to anyone who had already entered the University when they began. Their weighted still sums up to about 120 because the weighted connections in any course still sum up to approximately the number of credit hours the course offers. The weights of all connections in a given course are higher than they would have been if the missing enrollees were included.

Figure 12: Degree distributions of students by first year only including the years 2008-2010.

(a) A histogram showing the unweighted degree distribution of students segmented by first year from 2008-2010 "filtered" vs. all students "unfiltered".



(b) A histogram showing the weighted degree distribution of students segmented by first year from 2008-2010 "filtered" vs. all students "unfiltered".



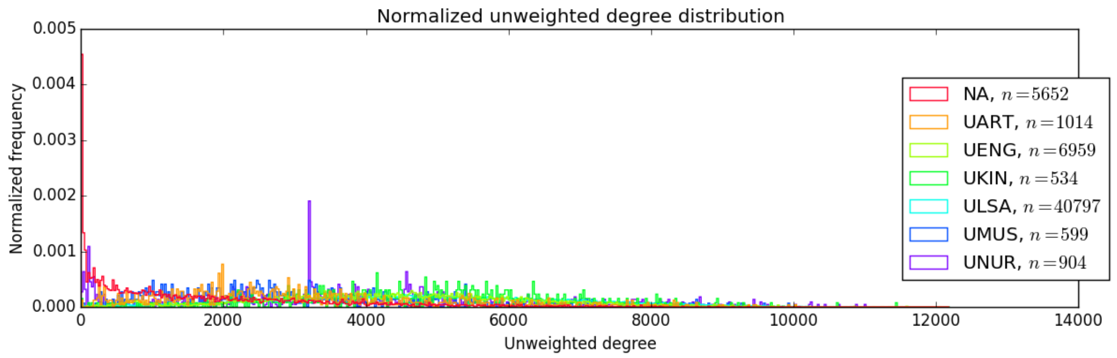
Now that we understand that missing connections affect the weighted and unweighted degree distributions, we may want to know what the distributions would look like if we excluded students who are missing a full set of connections. By filtering out students who did not start between the years of 2008 and 2010 and comparing them with all students, we arrive at the distributions shown in Figure 12. A comparison of the unweighted distributions in Figure 12a shows that the area between 0 and 4000 connections seems much more sparse, and the distribution appears to peak later than the unfiltered unweighted degree distribution. A KS test of the unfiltered distribution vs. the filtered distribution returns a KS statistic of 0.13, suggesting that they are noticeably different. The weighted distributions in Figure 12b is much more noticeably different. The large peaks before 100 have been significantly reduced, and the peak after 100 is much more pronounced. The KS test of the filtered vs. unfiltered distribution returns a KS statistic of 0.25.

3.2.2 Students segmented by school

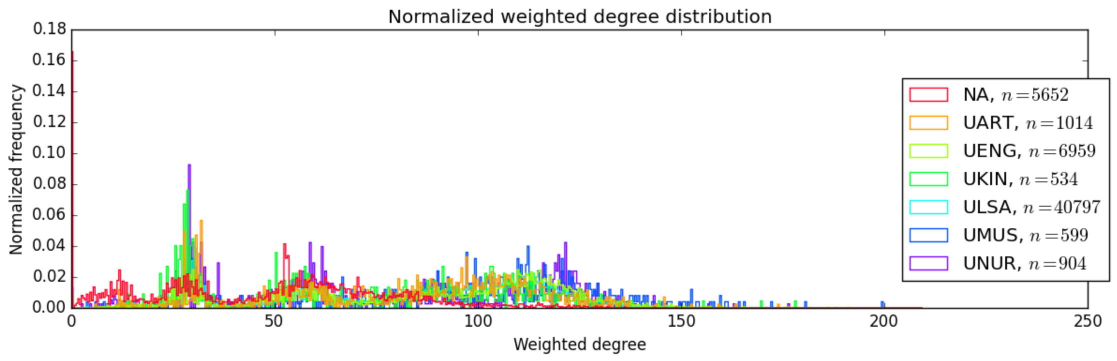
We also might wish to look at how students in different school differ in their connections. We provide weighted and unweighted distributions segmented by school in Figure 13.

Figure 13: Degree distributions of G_s. Note that in this visualization, we have removed several schools with relatively low enrollment rates to avoid visual clutter.

(a) A histogram showing the unweighted degree distribution of G_s segmented by school.



(b) A histogram showing the weighted degree distribution of G_s segmented by school.



The distributions in Figure 13 do suggest one interesting conclusion about our data: It appears that students whose school is classified as “NA” appear to be responsible for the population of students with zero connections. That said, beyond this observation, the distributions are mostly inscrutable, so we appeal to the KS heatmaps in Figure 14.

While there are a lot of “-” marks, Figure 14 does allows us to determine which pairs of schools we should inspect more closely. We return to the histogram of degree distribution visualization, but now focus our analysis on specific pairs of schools that look interesting.

Most striking are the differences in the rows marked “GRAC” (Graduate Rackham). We especially notice the high number in the cell corresponding to GRAC vs. PPHR (Pharmacy Professional). Unfortunately, while these distributions are different, the number of students in these programs is too small (PPHR has just 15 students) for a histogram to convey useful information. Instead, we concentrate attention on larger schools. For example, we can examine Music, Theatre, & Dance (UMUS) and Kinesiology (UKIN) . These are relatively larger schools, but they still have a KS statistic of 0.44 in unweighted degree distributions, and one of 0.26 in weighted degree distribution, so we should be able to spot some differences.

Indeed, we see a very clear difference. Consider Figure 15a. We see that the Kinesiology distribution peaks much later than Music, Theatre & Dance, suggesting that Music, Theatre & Dance students in general have fewer connections. However, Figure 15b makes sense if we recall that weighted degree is virtually equivalent to credit hours. Because both groups have to take about the same number of credits to graduate, their weighted distributions look similar.

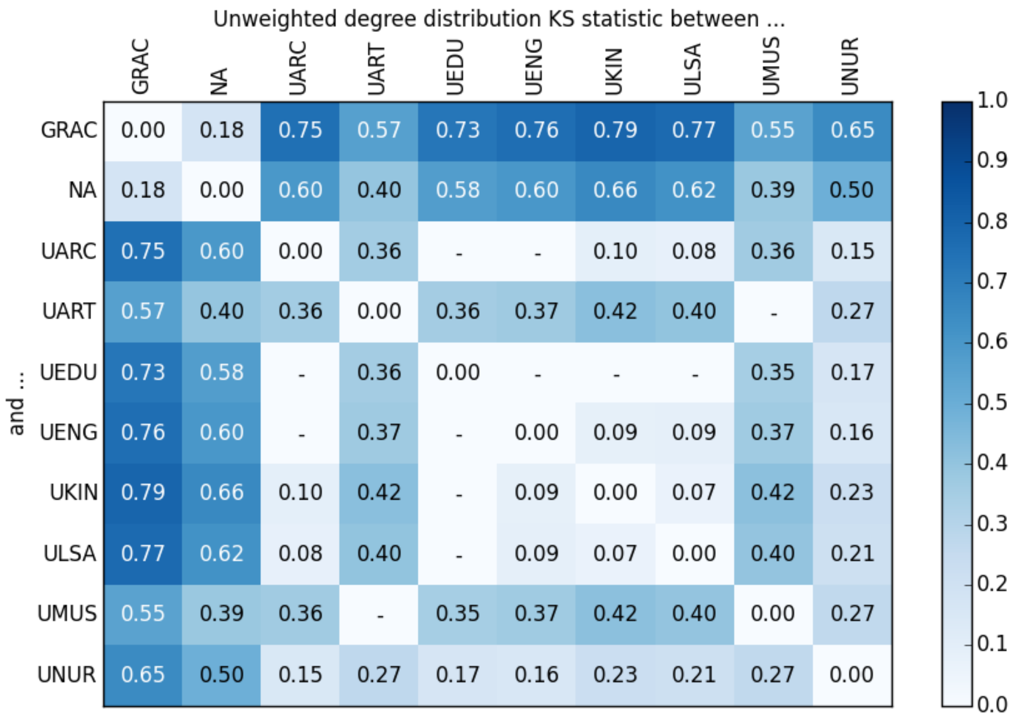
Another interesting comparison to make would be LSA vs. Engineering (ULSA vs. UENG), because they are the two largest schools in the data set, with 40797 and 5969 students respectively. We show these distributions in Figure 16. For the most part, differences between them are hard to spot. The Engineering unweighted distribution in Figure 16a clearly has more noise than the LSA distribution. It also has an interesting peak at the beginning around 1000 that the LSA distribution lacks. It also appears to peak slightly before the LSA distribution.

3.2.3 Students segmented by gender

Because gender is simply segmented into men and women, it’s relatively easy to analyze. KS statistics for gender are 0.02 for the unweighted distribution and 0.03 for the weighted distribution, implying that there is little discernible difference between the connections of the two populations. There isn’t much gained in an analysis of students segmented by gender beyond verifying that neither group shows unexpected patterns, so we do not provide any figures for this section.

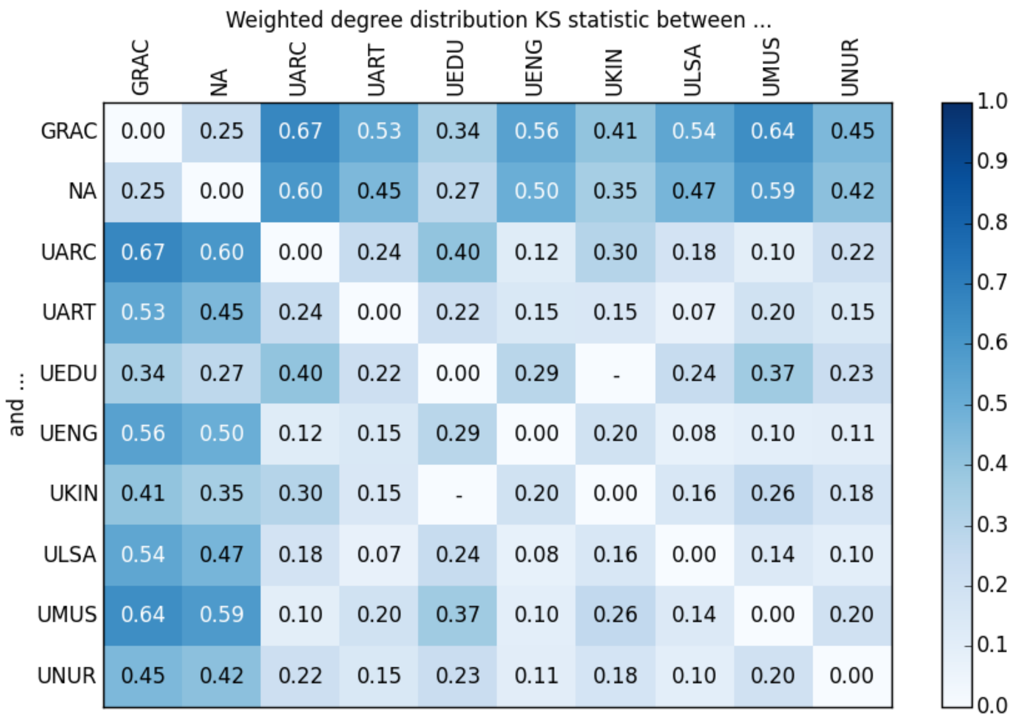
Figure 14: KS statistics for pairs of schools.

(a) A histogram showing the unweighted degree distribution of G_s segmented by school.



	GRAC	NA	UARC	UART	UEDU	UENG	UKIN	ULSA	UMUS	UNUR
GRAC	0.00	0.18	0.75	0.57	0.73	0.76	0.79	0.77	0.55	0.65
NA	0.18	0.00	0.60	0.40	0.58	0.60	0.66	0.62	0.39	0.50
UARC	0.75	0.60	0.00	0.36	-	-	0.10	0.08	0.36	0.15
UART	0.57	0.40	0.36	0.00	0.36	0.37	0.42	0.40	-	0.27
UEDU	0.73	0.58	-	0.36	0.00	-	-	-	0.35	0.17
UENG	0.76	0.60	-	0.37	-	0.00	0.09	0.09	0.37	0.16
UKIN	0.79	0.66	0.10	0.42	-	0.09	0.00	0.07	0.42	0.23
ULSA	0.77	0.62	0.08	0.40	-	0.09	0.07	0.00	0.40	0.21
UMUS	0.55	0.39	0.36	-	0.35	0.37	0.42	0.40	0.00	0.27
UNUR	0.65	0.50	0.15	0.27	0.17	0.16	0.23	0.21	0.27	0.00

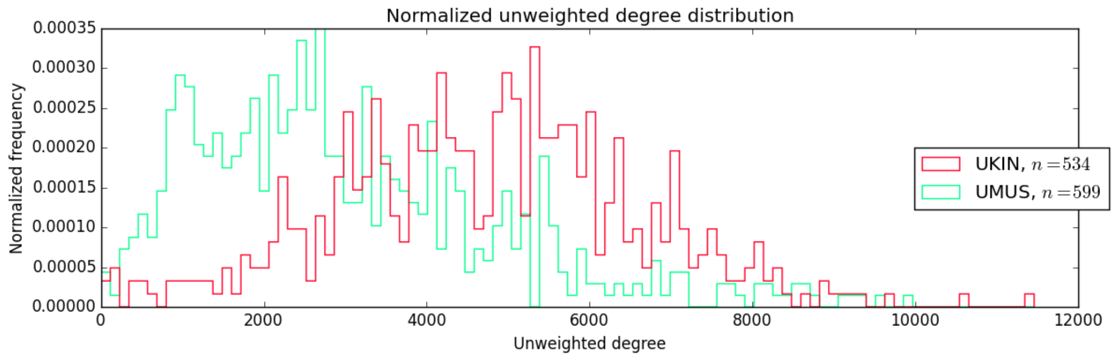
(b) A histogram showing the weighted degree distribution of G_s segmented by school.



	GRAC	NA	UARC	UART	UEDU	UENG	UKIN	ULSA	UMUS	UNUR
GRAC	0.00	0.25	0.67	0.53	0.34	0.56	0.41	0.54	0.64	0.45
NA	0.25	0.00	0.60	0.45	0.27	0.50	0.35	0.47	0.59	0.42
UARC	0.67	0.60	0.00	0.24	0.40	0.12	0.30	0.18	0.10	0.22
UART	0.53	0.45	0.24	0.00	0.22	0.15	0.15	0.07	0.20	0.15
UEDU	0.34	0.27	0.40	0.22	0.00	0.29	-	0.24	0.37	0.23
UENG	0.56	0.50	0.12	0.15	0.29	0.00	0.20	0.08	0.10	0.11
UKIN	0.41	0.35	0.30	0.15	-	0.20	0.00	0.16	0.26	0.18
ULSA	0.54	0.47	0.18	0.07	0.24	0.08	0.16	0.00	0.14	0.10
UMUS	0.64	0.59	0.10	0.20	0.37	0.10	0.26	0.14	0.00	0.20
UNUR	0.45	0.42	0.22	0.15	0.23	0.11	0.18	0.10	0.20	0.00

Figure 15: Degree distributions of students in Music, Theatre & Dance and Kinesiology.

(a) A histogram showing the unweighted degree distribution of students in Music, Theatre & Dance and Kinesiology.



(b) A histogram showing the weighted degree distribution of students in Music, Theatre & Dance and Kinesiology.

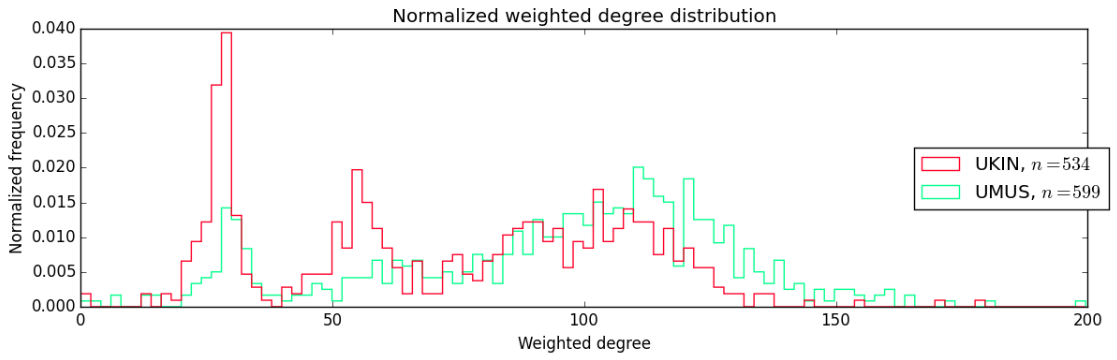
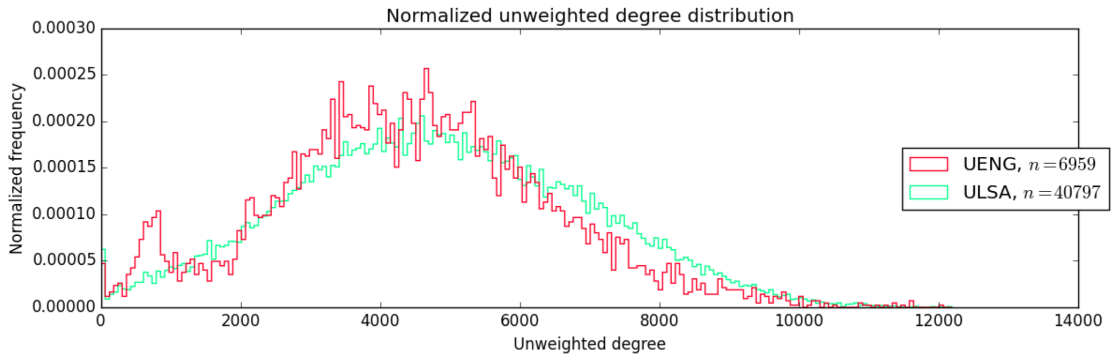


Figure 16: Degree distributions of students in LSA and Engineering.

(a) A histogram showing the unweighted degree distribution of students in LSA and Engineering.



(b) A histogram showing the weighted degree distribution of students in LSA and Engineering.

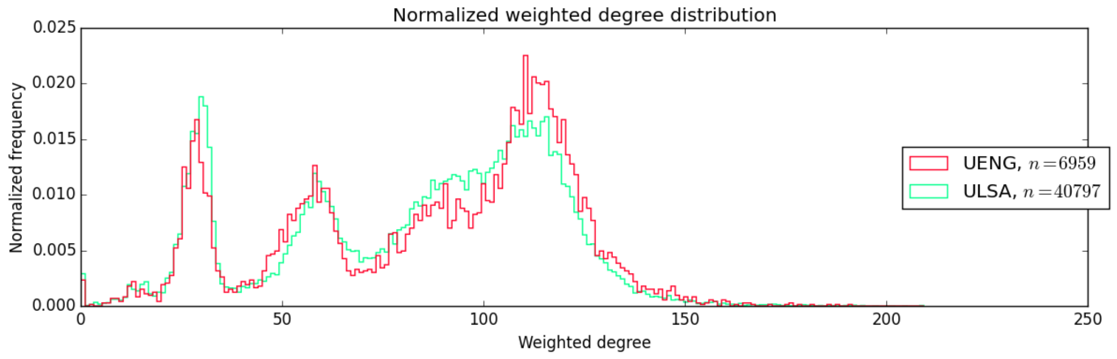
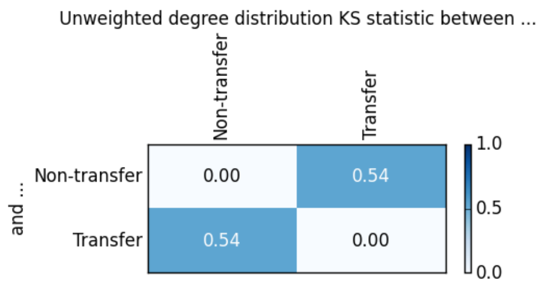
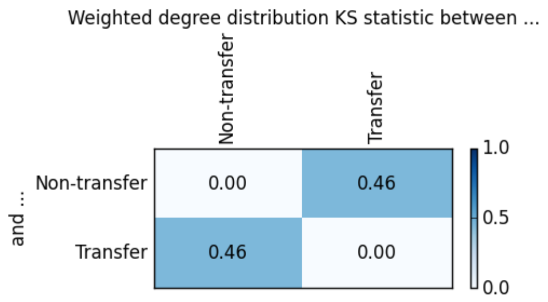


Figure 17: KS statistics for pairs of transfer statuses.



(a) A histogram showing the unweighted degree distribution of G_s segmented by transfer status.



(b) A histogram showing the weighted degree distribution of G_s segmented by transfer status.

3.2.4 Students segmented by transfer status

Because transfer students by definition come from other schools, we might expect that they have fewer connections than non-transfer students. Figure 17 does seem to indicate a difference.

The graphs of the degree distributions in Figure 18 confirm this difference. The unweighted degree distribution of transfer students appears to roughly decrease from 0, while the distribution for non-transfer students seems to smoothly curve up and peak between 4000 and 5000, then decrease. In the graph of weighted distribution, The distribution of transfer students appears to lack the peak at approximately 120 that non-transfer students have.

In general, this suggests that transfer students have connections to fewer students on campus than non-transfer students, and they have fewer credit hours. This makes sense; because transfer students take fewer classes, they make fewer connections. However, it is worth noting that this has practical significance; it means that the transfer student experience is much more lightly connected than students who are here a full four years.

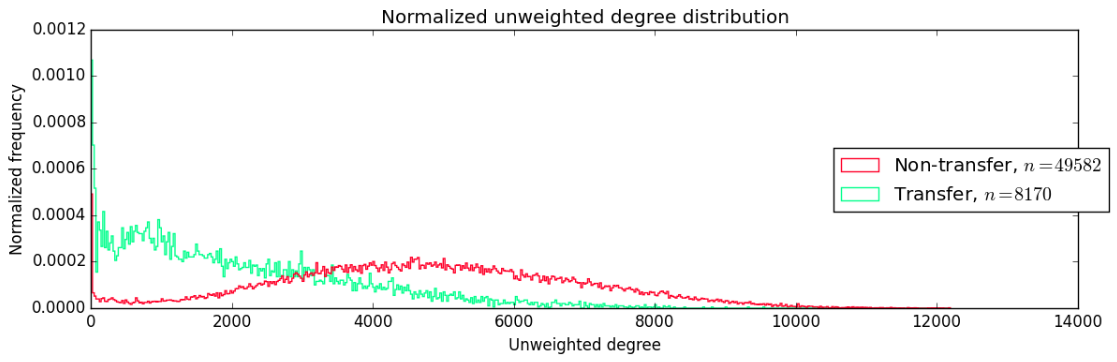
3.2.5 Students segmented by ethnicity

We'll also look at segmentation by ethnicity, beginning again with the KS heatmap in Figure 19.

While most of these distributions seem very similar based on the heatmap, we'll take a look at the distributions of Nonresident Alien vs. Asian only, the highest KS-statistic in both Figures 19b and 19a. Indeed, Figure 20b shows that the Nonresident Alien distribution has a peak around 10-15 that the Asian only distribution lacks, and the Asian only distribution has a higher peak at 120. Figure 20a shows a difference between the distributions not unlike that of transfer vs. non-transfer in Figure 18a. This is an interesting note, Nonresident Aliens students appear to have fewer connections at this University than Asian only students,

Figure 18: Degree distributions of transfer and non-transfer students.

(a) A histogram showing the unweighted degree distribution of transfer and non-transfer students.



(b) A histogram showing the weighted degree distribution of transfer and non-transfer students.

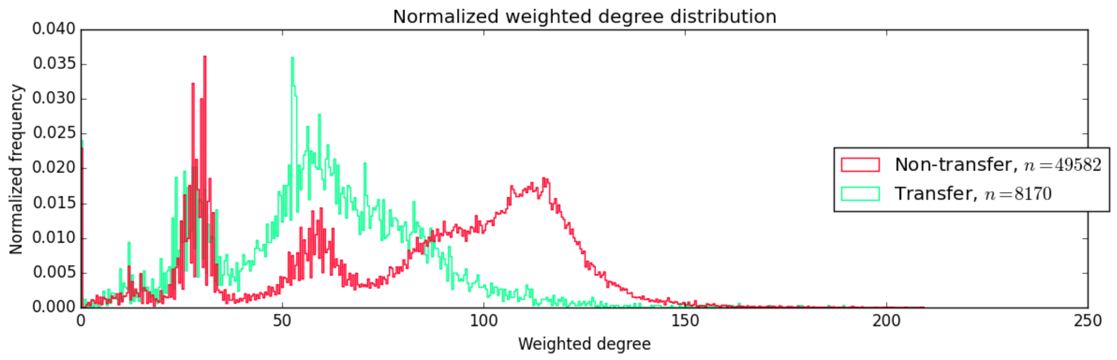
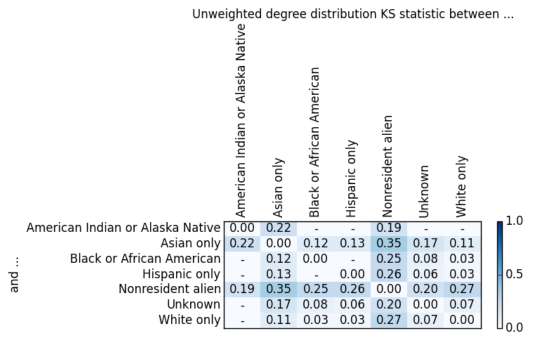


Figure 19: KS statistics for pairs of first years.

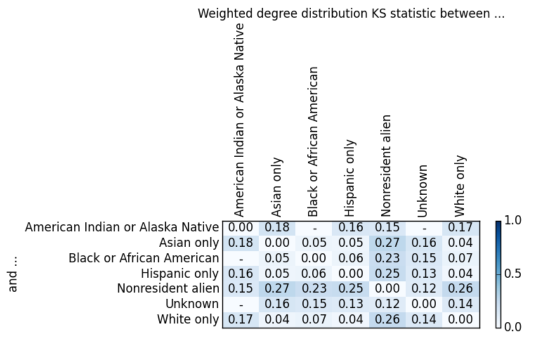
(a) A histogram showing the unweighted degree distribution of G_s segmented by ethnicity.



Unweighted degree distribution KS statistic between ...

and ...	American Indian or Alaska Native	Asian only	Black or African American	Hispanic only	Nonresident alien	Unknown	White only
American Indian or Alaska Native	0.00	0.22	-	-	0.19	-	-
Asian only	0.22	0.00	0.12	0.13	0.35	0.17	0.11
Black or African American	-	0.12	0.00	-	0.25	0.08	0.03
Hispanic only	-	0.13	-	0.00	0.26	0.06	0.03
Nonresident alien	0.19	0.35	0.25	0.26	0.00	0.20	0.27
Unknown	-	0.17	0.08	0.06	0.20	0.00	0.07
White only	-	0.11	0.03	0.03	0.27	0.07	0.00

(b) A histogram showing the weighted degree distribution of G_s segmented by ethnicity.

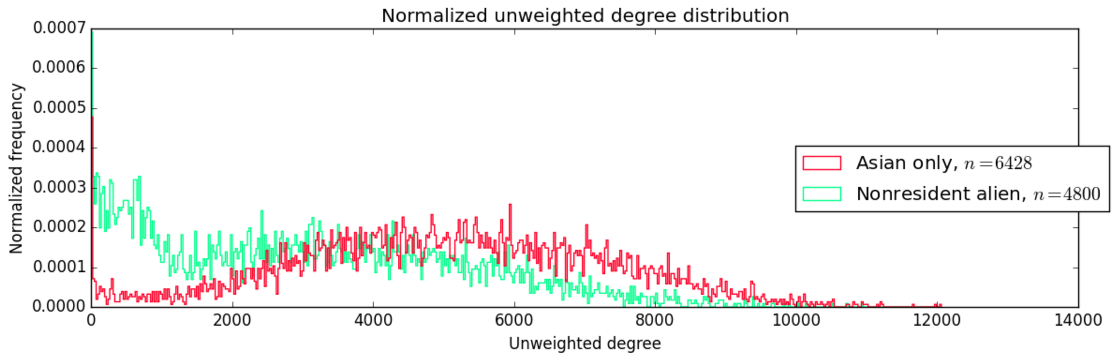


Weighted degree distribution KS statistic between ...

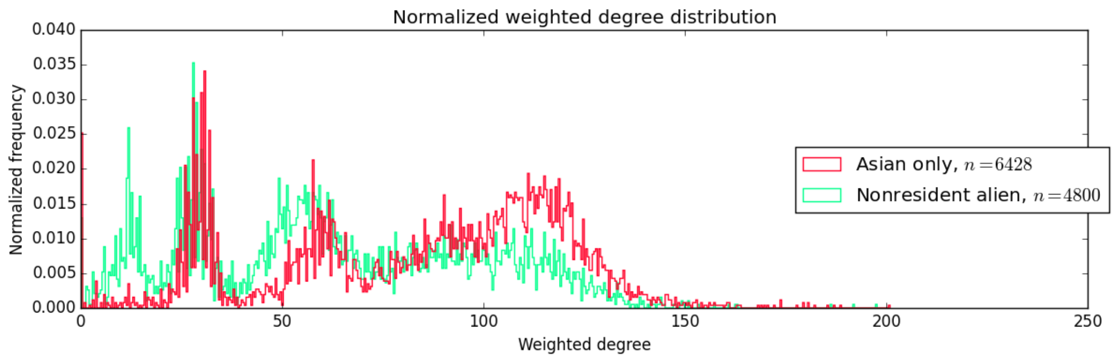
and ...	American Indian or Alaska Native	Asian only	Black or African American	Hispanic only	Nonresident alien	Unknown	White only
American Indian or Alaska Native	0.00	0.18	-	0.16	0.15	-	0.17
Asian only	0.18	0.00	0.05	0.05	0.27	0.16	0.04
Black or African American	-	0.05	0.00	0.06	0.23	0.15	0.07
Hispanic only	0.16	0.05	0.06	0.00	0.25	0.13	0.04
Nonresident alien	0.15	0.27	0.23	0.25	0.00	0.12	0.26
Unknown	-	0.16	0.15	0.13	0.12	0.00	0.14
White only	0.17	0.04	0.07	0.04	0.26	0.14	0.00

Figure 20: Degree distributions of Nonresident Alien vs. Asian only students.

(a) A histogram showing the unweighted degree distribution of Nonresident Alien vs. Asian only students.



(b) A histogram showing the weighted degree distribution of Nonresident Alien vs. Asian only students.



who do not seem to be different from other ethnicities. Perhaps many Nonresident Alien students are transfer students. Indeed, when we check the numbers, we find that about 40% of Nonresident Alien students are transfer students, while only about 14% of the entire data set consists of transfer students.

3.2.6 Students segmented by major

The final segmentation scheme we examine is students segmented by major. This is slightly more complex than previous segmentation schemes because there are many majors. Unfortunately, the KS heatmap is too large to display on this page. Instead, we forgo the visualization and find the pairs of majors with the largest KS statistic.

The top ten highest and lowest KS-statistics for majors with over 100 students in the data set with $p < 0.05$ are included in Table 4.

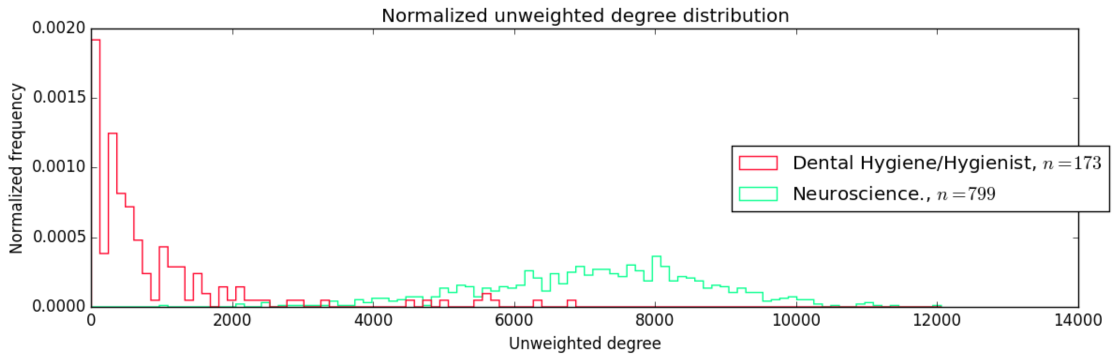
Because Dental Hygiene seems to appear often with a large KS statistic, it appears that Dental Hygiene has a different distribution than other majors. Figure 21 shows the distributions of Dental Hygiene vs. Neuroscience. On the low end of the distributions, we have pairs of

Table 4: Pairs of majors and the corresponding KS statistics of their degree distributions with over 100 students in the data where $p < 0.05$

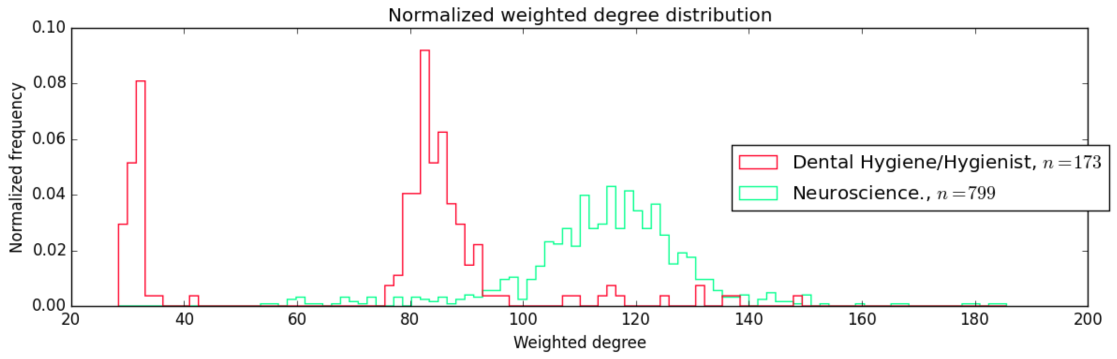
Major 1	**Major 2**	**KS statistic**
		Unweighted
Dental Hygiene/Hygienist	Neuroscience	0.93
Organizational Behavior Studies	Dental Hygiene/Hygienist	0.92
Ecology and Evolutionary Biology	Dental Hygiene/Hygienist	0.92
Dental Hygiene/Hygienist	Physiological Psychology/Psychology	0.91
Dental Hygiene/Hygienist	Microbiology, General	0.91
Dental Hygiene/Hygienist	Cell/Cellular and Molecular Biology	0.90
Dental Hygiene/Hygienist	Public Policy Analysis	0.90
Music Performance, General	Neuroscience.	0.90
Dental Hygiene/Hygienist	Biology/Biological Sciences, General	0.90
Dental Hygiene/Hygienist	Biochemistry	0.90
Mechanical Engineering	Electrical, Electronics and Computer Engineering	0.10
Communication Studies/ Speech Communication	Anthropology	0.10
Mathematics, General	English Language and Literature	0.10
Mathematics, General	History, General	0.10
Sociology	Political Science and Government	0.10
Economics, General	Political Science and Government	0.09
Political Science and Government	Mathematics, General	0.08
Industrial Engineering	English Language and Literature	0.08
Business Administration and Marketing	Communication Studies/ Speech Communication	0.08
Chemical Engineering	Economics, General	0.07
		Weighted
Dental Hygiene/Hygienist	Neuroscience	0.85
Dental Hygiene/Hygienist	Microbiology, General	0.85
Dental Hygiene/Hygienist	Informatics	0.84
Dental Hygiene/Hygienist	Physiological Psychology/Psychology	0.81
Dental Hygiene/Hygienist	Kinesiology and Exercise Science	0.79
Dental Hygiene/Hygienist	Chemical Engineering	0.78
Dental Hygiene/Hygienist	Cell/Cellular and Molecular Biology	0.78
Business Administration and Marketing	Informatics	0.78
Business Administration and Marketing	Microbiology, General	0.78
Business Administration and Marketing	Neuroscience	0.77
Cell/Cellular and Molecular Biology	Industrial Engineering	0.09
Political Science and Government	Sport and Fitness Administration	0.09
Aerospace, Aeronautical	Industrial Engineering	0.09
Computer and Information Science	Industrial Engineering	0.08
Environmental Studies	Political Science and Government	0.08
Political Science and Government	English Language and Literature	0.08
Experimental Psychology	Mathematics, General	0.08
Political Science and Government	History, General	0.08
Political Science and Government	Experimental Psychology	0.08
Political Science and Government	Economics, General	0.06

Figure 21: Degree distributions of Dental Hygiene vs. Neuroscience students.

(a) A histogram showing the unweighted degree distribution of Dental Hygiene vs. Neuroscience students. KS statistic: 0.93



(b) A histogram showing the weighted degree distribution of Dental Hygiene vs. Neuroscience students. KS statistic: 0.85



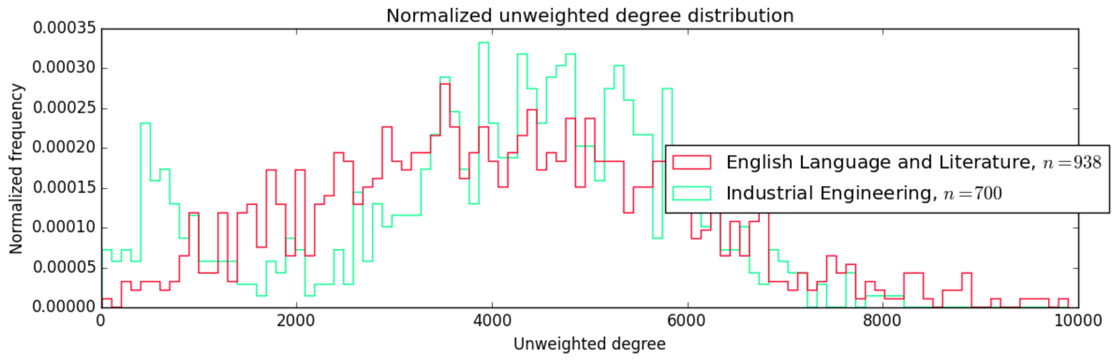
majors who are generally in the same school, even though the content may be different, like English and Math or Mechanical Engineering and Electrical/Computer Engineering. However, we even see pairs like Chemical Engineering and Economics, because, as we established in Section 3.2.2, LSA and Engineering students do not have distributions of connections that are very different. Figure 22 shows the degree distributions of Industrial Engineering vs. English students.

Based on Figure 21, Dental Hygiene students appear to be a close-knit community that takes most of their classes together. The unweighted distribution (Figure 21a) shows a small number of connections, but the weighted distribution (Figure 21b) shows a large peak before from the peak of Neuroscience students. Perhaps the Dental Hygiene program does not require as many credit hours as other programs.

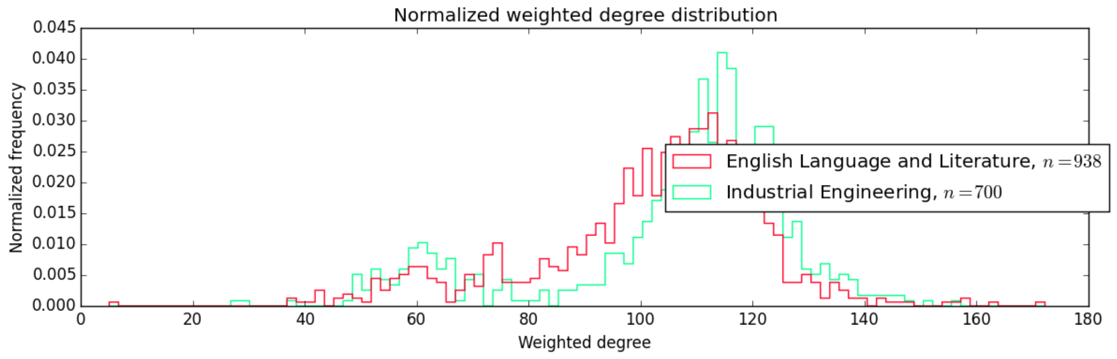
The degree distributions in Figure 22 are more similar, but there are certainly noticeable differences. Industrial Engineering students seem to have a peak early in the unweighted distribution.

Figure 22: Degree distributions of Industrial Engineering vs. English students.

(a) A histogram showing the unweighted degree distribution of Industrial Engineering vs. English students. KS statistic: 0.08



(b) A histogram showing the weighted degree distribution of Industrial Engineering vs. English students. KS statistic: 0.18



3.3 Interactions between various segments of students

While we have examined how connected each segment of students is overall, one of our goals with this thesis was to be able to assess the richness of a student's curriculum. Beyond just the number of connections that students have, we'd like to be able to understand who they are connected to. Is their pool of connections homogeneous, or does it represent a more diverse community? Again, we use the idea of segmentation, but now, we look at interactions between various segments to see how they compare to expected interactions given uniformly distributed students.

Before we continue, we give a more rigorous definition of "expected interactions". Essentially, we wish to compare how much interaction groups of students actually had with what they might have had on a random graph. Like Section 3.2, we wish to do this in both an unweighted and weighted manner. We consider two ways of defining expected number of interactions. Let's define expected number of unweighted interactions from segment s_i to s_j as $\mathbb{E}[I_u(s_i, s_j)]$, and expected number of weighted interactions from segment s_i to s_j as $\mathbb{E}[I_w(s_i, s_j)]$.

1. The first way involves calculating an expected number of interactions between segments per course, then taking the summation over all courses. Given that there are $n_{s_i,c}$ and $n_{s_j,c}$ people in segments s_i and s_j in course c, we'd expect the number of connections in the unweighted graph to be: $n_{s_i,c} \times n_{s_j,c}$. Because each connection also has weight of 1, $n_{s_i,c} \times n_{s_j,c}$ would also be the connection strength between segments s_i and s_j in course c. In the weighted graph, the number of connections would be the same, but the strength of each connection would depend on the number of credits H_c and the size of enrollment $|E_c|$: $H_c/|E_c|$. Then, the overall connection strength between segments s_i and s_j would be $H_c/|E_c| \times n_{s_i,c} \times n_{s_j,c}$. Now, we compute the expected summation of interactions over all courses for segments s_i and s_j. Before we start, it's worth nothing that we can calculate the $n_{s_i,c} = |E_c| \times n_{s_i}/n$, where n_{s_i} is the total of students in segment s_i and n is the total number of students. For the unweighted graph:

$$\begin{aligned}
\mathbb{E}[I_u(s_i, s_j)] &= \sum_{c\in C} n_{s_i,c} \times n_{s_j,c} \\
&= \sum_{c\in C} \left(|E_c| \times \frac{n_{s_i}}{n}\right) \times \left(|E_c| \times \frac{n_{s_j}}{n}\right) \\
&= \sum_{c\in C} |E_c|^2 \times \frac{n_{s_i}}{n} \times \frac{n_{s_j}}{n} \\
&= \frac{n_{s_i}}{n} \times \frac{n_{s_j}}{n} \sum_{c\in C} |E_c|^2 \\
&= \frac{n_{s_i} \times n_{s_j}}{n^2} \sum_{c\in C} |E_c|^2
\end{aligned}$$

For the weighted graph:

$$\begin{aligned}
\mathbb{E}[I_w(s_i, s_j)] &= \sum_{c \in C} \frac{H_c}{|E_c|} \times n_{s_i,c} \times n_{s_j,c} \\
&= \sum_{c \in C} \frac{H_c}{|E_c|} \left(|E_c| \times \frac{n_{s_i}}{n}\right) \times \left(|E_c| \times \frac{n_{s_j}}{n}\right) \\
&= \sum_{c \in C} (|E_c| \times H_c) \times \left(\frac{n_{s_i}}{n} \times \frac{n_{s_j}}{n}\right) \\
&= \frac{n_{s_i}}{n} \times \frac{n_{s_j}}{n} \sum_{c \in C} |E_c| \times H_c \\
&= \frac{n_{s_i} \times n_{s_j}}{n^2} \sum_{c \in C} |E_c| \times H_c
\end{aligned}$$

Upon doing analysis with this method, however, it does not yield results that were useful. The underlying issue with this method is that it assumes students in different segments take courses at the same rate. If students from one segment take classes at a lower rate, then the actual number of interactions with that segment will be lower than the expected number of interactions every time. This does convey some information, but we'd prefer to see, given the number of interactions, which pairs of segments have greater number of interactions, and which have fewer than they would at random. This method does not fix enough parameters in our random graph.

2. We fix the number of interactions in our second method. This method of calculating expected interactions is much simpler. It starts with a given segment s_i and the number of unweighted connections $I_{u,i}$ and weighted connections $I_{w,i}$ for s_i. Then, it says that the number of connections between s_i and s_j should be:

$$\mathbb{E}[I_x(s_i, s_j)] = I_{x,i} \times \frac{n_{s_j}}{n} \tag{3}$$

I_x is either I_u or I_w. Colloquially, the formula says that the proportion of the total connections s_i has with s_j is proportional to the number of students in s_j. This method of calculating expected interactions is remarkably simpler to state, and meets the criterion that some interactions should be greater than expected and some less for a given segment. In fact, the differences between actual and expected number of interactions is zero sum.

One final mathematical concept remains. While we can now take the difference between the actual interactions between two segments and their expected interactions, the difference depends on the size of the segments, to fix this, we compute the relative percent difference (RPD), defined in (4).

$$\text{relative percent difference} = \frac{\text{actual connections} - \text{expected connections}}{(\text{actual connections} + \text{expected connections})/2} \tag{4}$$

Put more simply, the RPD is the difference over the average. It normalizes for segment sizes.

3.3.1 Students segmented by school

To visualize the interactions between segments of students we employ the use of a chord diagram, a way to represent a matrix of values where the row and column labels are placed around a circle, and the values in corresponding cells are indicated by the size of the chord between two labels. We present the chord diagram for interactions between schools in Figure 23.

While this diagram gives us an idea of the scale of the interactions between various segments of students, like the histograms in Section 3.2, it does not give us an idea of what expected interactions should be. Again, we present heatmaps to give us an idea of where we should draw our attention. This time, our heatmaps present RPD between actual and expected interactions.

We present a few notes on how to interpret Figure 24. The matrices presented are not symmetric. While the number of interactions between any two segments is symmetric, recall that our method of calculating expected interactions, given by (3), is not. To get a sense of segment s_i's interactions with other segments, look down the column labeled s_i. This gives the difference of interactions from s_i to s_j. To get a sense of the isolation or connectedness of segment s_i, look at the row marked s_i. The row contains information on how the segment s_i receives connections from other segments s_j.

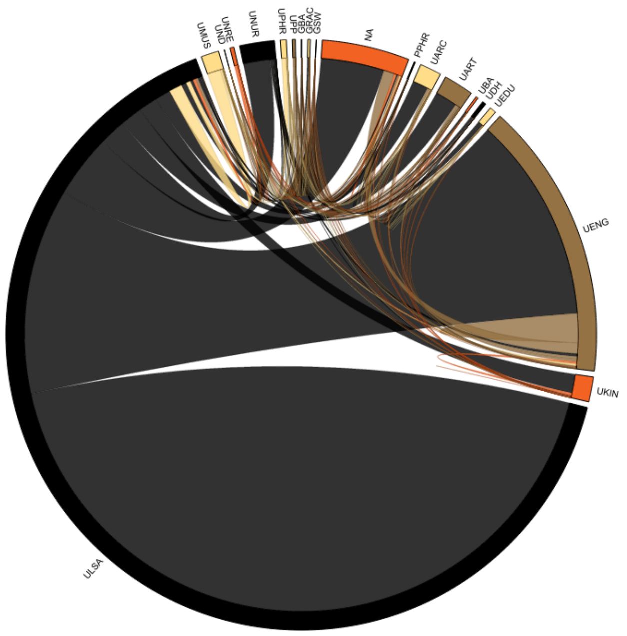
Most striking about the matrices in Figure 24 is the isolation of graduate students in Rackham (GRAC). Most other schools have far fewer interactions with it than one would expected based on its number of students. Even given its own set of interactions, it appears to only be connected to Professional Pharmacy (PPHR), Engineering (UENG), and itself. Contrast this with LSA (ULSA), which, as the largest school in the data set and likely home to the most diverse set of student interests, is well connected to almost every other school. Somewhwere in the middle is Engineering, which appears to be more connected than Rackham, but is not connected to some of the more specialized schools, such as Kinesiology (UKIN), Education (UEDU), Music (UMUS), and Art (UART).

3.3.2 Students segmented by ethnicity

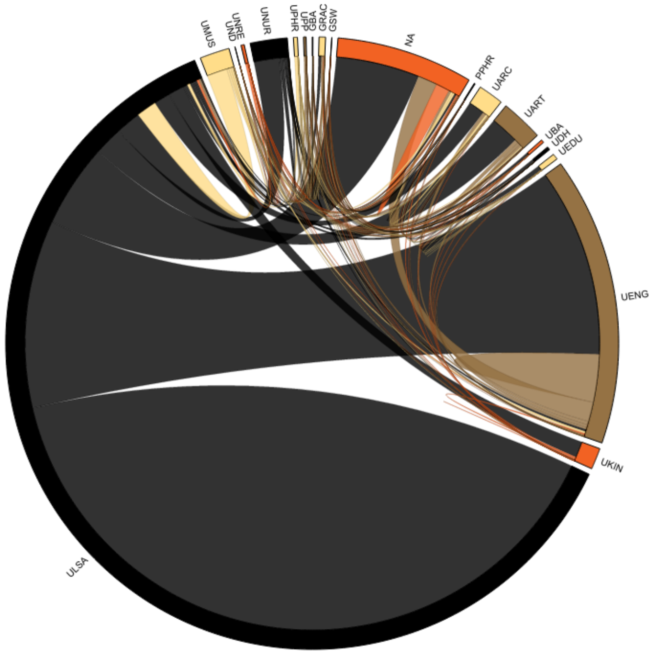
We first present the chord diagrams and interaction RPD heatmaps in Figures 25 and 26.

A few comments can be made about the relative percent differences between actual and expected interactions as indicated by Figure 26. In the unweighted heatmap (Figure 26a) most striking is the negative values along the diagonal. This suggests that students of a given ethnicity do not have as many connections to others of their own ethnicity as we would expect. When we account for weight (Figure 26b), we see a similar story, with the exception of Black or African American students, who have a higher interaction strength than expected. This is an interesting exception to the trend, and may result from the fact that students from historically marginalized communities have a tendency to stick together. Sociological reasons for this have been documented in other research [31]. We see the same

Figure 23: Chord diagrams of interactions between students segmented by school.



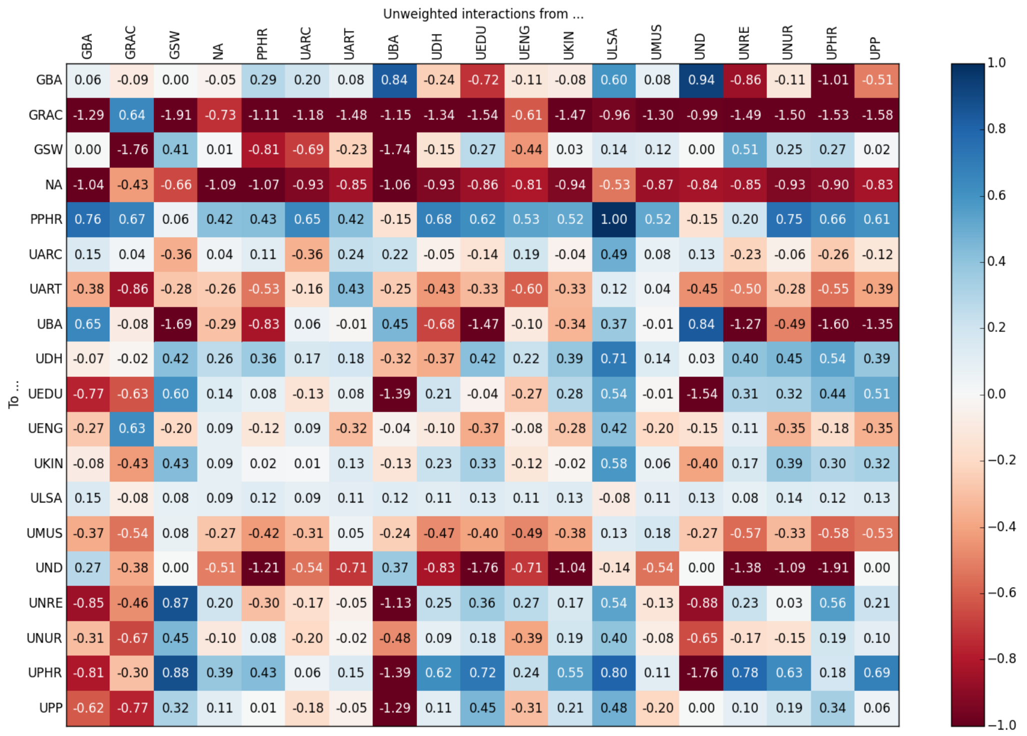
(a) Chord diagram of unweighted interactions between students segmented by school.



(b) Chord diagram of weighted interactions between students segmented by school.

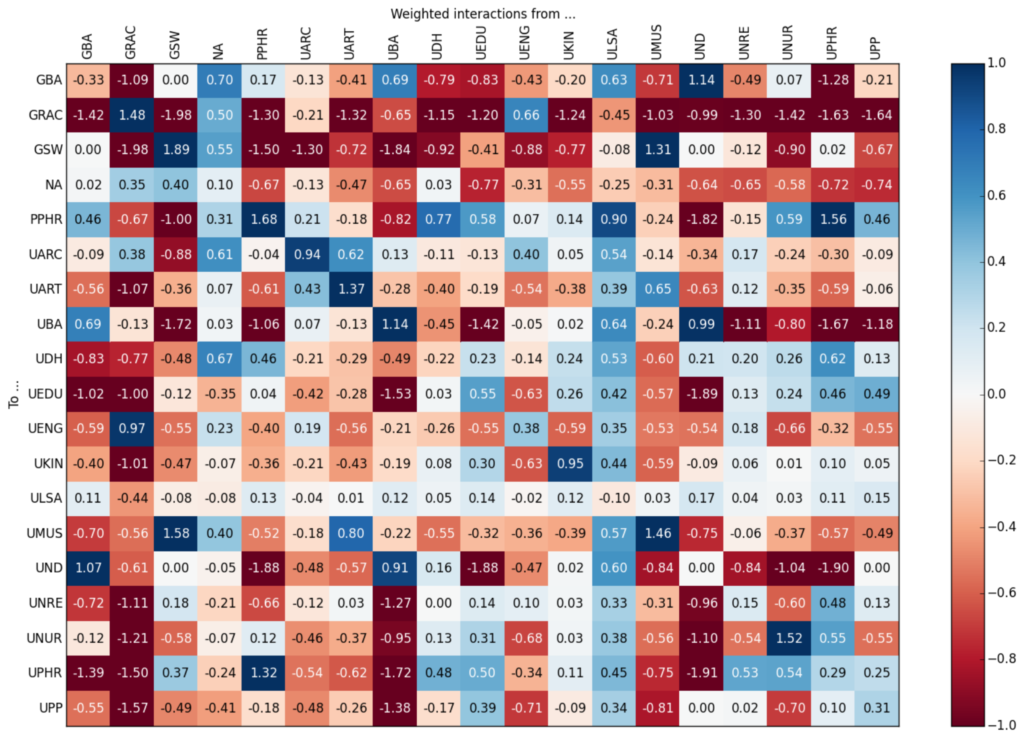
Figure 24: RPD between actual and expected interactions.

(a) RPD between actual and expected unweighted interactions of students segmented by school.



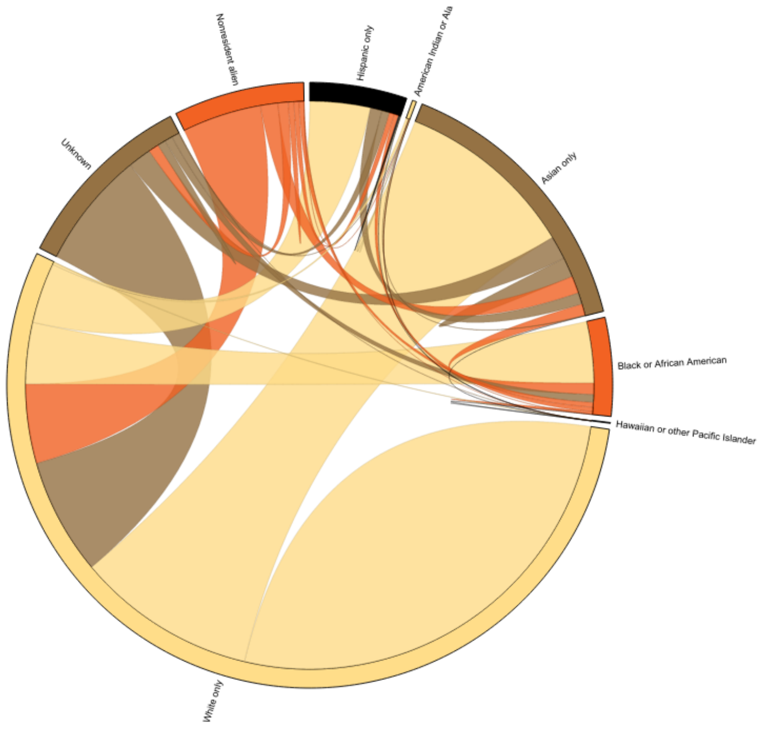
To ... \ from ...	GBA	GRAC	GSW	NA	PPHR	UARC	UART	UBA	UDH	UEDU	UENG	UKIN	ULSA	UMUS	UND	UNRE	UNUR	UPHR	UPP
GBA	0.06	-0.09	0.00	-0.05	0.29	0.20	0.08	0.84	-0.24	-0.72	-0.11	-0.08	0.60	0.08	0.94	-0.86	-0.11	-1.01	-0.51
GRAC	-1.29	0.64	-1.91	-0.73	-1.11	-1.18	-1.48	-1.15	-1.34	-1.54	-0.61	-1.47	-0.96	-1.30	-0.99	-1.49	-1.50	-1.53	-1.58
GSW	0.00	-1.76	0.41	0.01	-0.81	-0.69	-0.23	-1.74	-0.15	0.27	-0.44	0.03	0.14	0.12	0.00	0.51	0.25	0.27	0.02
NA	-1.04	-0.43	-0.66	-1.09	-1.07	-0.93	-0.85	-1.06	-0.93	-0.86	-0.81	-0.94	-0.53	-0.87	-0.84	-0.85	-0.93	-0.90	-0.83
PPHR	0.76	0.67	0.06	0.42	0.43	0.65	0.42	-0.15	0.68	0.62	0.53	0.52	1.00	0.52	-0.15	0.20	0.75	0.66	0.61
UARC	0.15	0.04	-0.36	0.04	0.11	-0.36	0.24	0.22	-0.05	-0.14	0.19	-0.04	0.49	0.08	0.13	-0.23	-0.06	-0.26	-0.12
UART	-0.38	-0.86	-0.28	-0.26	-0.53	-0.16	0.43	-0.25	-0.43	-0.33	-0.60	-0.33	0.12	0.04	-0.45	-0.50	-0.28	-0.55	-0.39
UBA	0.65	-0.08	-1.69	-0.29	-0.83	0.06	-0.01	0.45	-0.68	-1.47	-0.10	-0.34	0.37	-0.01	0.84	-1.27	-0.49	-1.60	-1.35
UDH	-0.07	-0.02	0.42	0.26	0.36	0.17	0.18	-0.32	-0.37	0.42	0.22	0.39	0.71	0.14	0.03	0.40	0.45	0.54	0.39
UEDU	-0.77	-0.63	0.60	0.14	0.08	-0.13	0.08	-1.39	0.21	-0.04	-0.27	0.28	0.54	-0.01	-1.54	0.31	0.32	0.44	0.51
UENG	-0.27	0.63	-0.20	0.09	-0.12	0.09	-0.32	-0.04	-0.10	-0.37	-0.08	-0.28	0.42	-0.20	-0.15	0.11	-0.35	-0.18	-0.35
UKIN	-0.08	-0.43	0.43	0.09	0.02	0.01	0.13	-0.13	0.23	0.33	-0.12	-0.02	0.58	0.06	-0.40	0.17	0.39	0.30	0.32
ULSA	0.15	-0.08	0.08	0.09	0.12	0.09	0.11	0.12	0.11	0.13	0.11	0.13	-0.08	0.11	0.13	0.08	0.14	0.12	0.13
UMUS	-0.37	-0.54	0.08	-0.27	-0.42	-0.31	0.05	-0.24	-0.47	-0.40	-0.49	-0.38	0.13	0.18	-0.27	-0.57	-0.33	-0.58	-0.53
UND	0.27	-0.38	0.00	-0.51	-1.21	-0.54	-0.71	0.37	-0.83	-1.76	-0.71	-1.04	-0.14	-0.54	0.00	-1.38	-1.09	-1.91	0.00
UNRE	-0.85	-0.46	0.87	0.20	-0.30	-0.17	-0.05	-1.13	0.25	0.36	0.27	0.17	0.54	-0.13	-0.88	0.23	0.03	0.56	0.21
UNUR	-0.31	-0.67	0.45	-0.10	0.08	-0.20	-0.02	-0.48	0.09	0.18	-0.39	0.19	0.40	-0.08	-0.65	-0.17	-0.15	0.19	0.10
UPHR	-0.81	-0.30	0.88	0.39	0.43	0.06	0.15	-1.39	0.62	0.72	0.24	0.55	0.80	0.11	-1.76	0.78	0.63	0.18	0.69
UPP	-0.62	-0.77	0.32	0.11	0.01	-0.18	-0.05	-1.29	0.11	0.45	-0.31	0.21	0.48	-0.20	0.00	0.10	0.19	0.34	0.06

(b) RPD between actual and expected weighted interactions of students segmented by school.

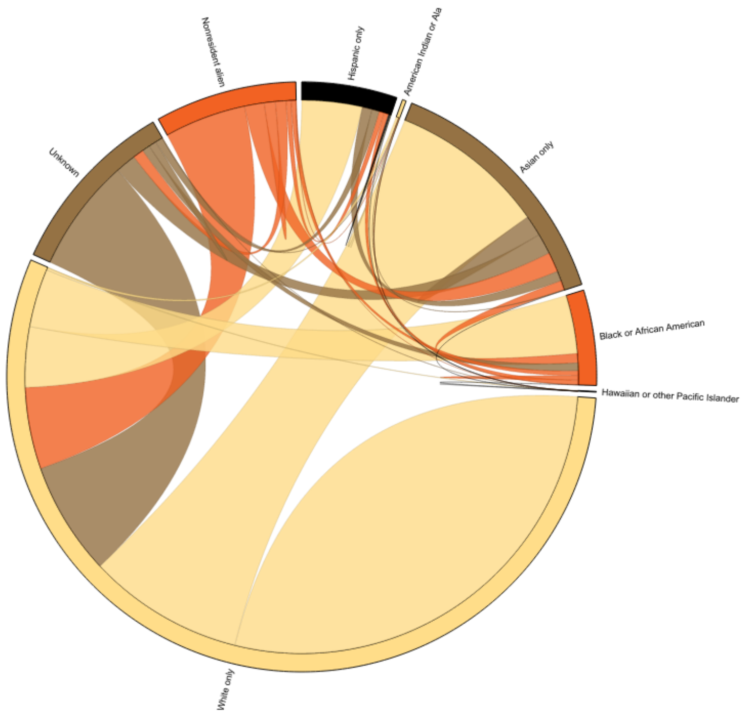


To ... \ from ...	GBA	GRAC	GSW	NA	PPHR	UARC	UART	UBA	UDH	UEDU	UENG	UKIN	ULSA	UMUS	UND	UNRE	UNUR	UPHR	UPP
GBA	-0.33	-1.09	0.00	0.70	0.17	-0.13	-0.41	0.69	-0.79	-0.83	-0.43	-0.20	0.63	-0.71	1.14	-0.49	0.07	-1.28	-0.21
GRAC	-1.42	1.48	-1.98	0.50	-1.30	-0.21	-1.32	-0.65	-1.15	-1.20	0.66	-1.24	-0.45	-1.03	-0.99	-1.30	-1.42	-1.63	-1.64
GSW	0.00	-1.98	1.89	0.55	-1.50	-1.30	-0.72	-1.84	-0.92	-0.41	-0.88	-0.77	-0.08	1.31	0.00	-0.12	-0.90	0.02	-0.67
NA	0.02	0.35	0.40	0.10	-0.67	-0.13	-0.47	-0.65	0.03	-0.77	-0.31	-0.55	-0.25	-0.31	-0.64	-0.65	-0.58	-0.72	-0.74
PPHR	0.46	-0.67	-1.00	0.31	1.68	0.21	-0.18	-0.82	0.77	0.58	0.07	0.14	0.90	-0.24	-1.82	-0.15	0.59	1.56	0.46
UARC	-0.09	0.38	-0.88	0.61	-0.04	0.94	0.62	0.13	-0.11	-0.13	0.40	0.05	0.54	-0.14	-0.34	0.17	-0.24	-0.30	-0.09
UART	-0.56	-1.07	-0.36	0.07	-0.61	0.43	1.37	-0.28	-0.40	-0.19	-0.54	-0.38	0.39	0.65	-0.63	0.12	-0.35	-0.59	-0.06
UBA	0.69	-0.13	-1.72	0.03	-1.06	0.07	-0.13	1.14	-0.45	-1.42	-0.05	0.02	0.64	-0.24	0.99	-1.11	-0.80	-1.67	-1.18
UDH	-0.83	-0.77	-0.48	0.67	0.46	-0.21	-0.29	-0.49	-0.22	0.23	-0.14	0.24	0.53	-0.60	0.21	0.20	0.26	0.62	0.13
UEDU	-1.02	-1.00	-0.12	-0.35	0.04	-0.42	-0.28	-1.53	0.03	0.55	-0.63	0.26	0.42	-0.57	-1.89	0.13	0.24	0.46	0.49
UENG	-0.59	0.97	-0.55	0.23	-0.40	0.19	-0.56	-0.21	-0.26	-0.55	0.38	-0.59	0.35	-0.53	-0.54	0.18	-0.66	-0.32	-0.55
UKIN	-0.40	-1.01	-0.47	-0.07	-0.36	-0.21	-0.43	-0.19	0.08	0.30	-0.63	0.95	0.44	-0.59	-0.09	0.06	0.01	0.10	0.05
ULSA	0.11	-0.44	-0.08	-0.08	0.13	-0.04	0.01	0.12	0.05	0.14	-0.02	0.12	-0.10	0.03	0.17	0.04	0.03	0.11	0.15
UMUS	-0.70	-0.56	1.58	0.40	-0.52	-0.18	0.80	-0.22	-0.55	-0.32	-0.36	-0.39	0.57	1.46	-0.75	-0.06	-0.37	-0.57	-0.49
UND	1.07	-0.61	0.00	-0.05	-1.88	-0.48	-0.57	0.91	0.16	-1.88	-0.47	0.02	0.60	-0.84	0.00	-0.84	-1.04	-1.90	0.00
UNRE	-0.72	-1.11	0.18	-0.21	-0.66	-0.12	0.03	-1.27	0.00	0.14	0.10	0.03	0.33	-0.31	-0.96	0.15	-0.60	0.48	0.13
UNUR	-0.12	-1.21	-0.58	-0.07	0.12	-0.46	-0.37	-0.95	0.13	0.31	-0.68	0.03	0.38	-0.56	-1.10	-0.54	1.52	0.55	-0.55
UPHR	-1.39	-1.50	0.37	-0.24	1.32	-0.54	-0.62	-1.72	0.48	0.50	-0.34	0.11	0.45	-0.75	-1.91	0.53	0.54	0.29	0.25
UPP	-0.55	-1.57	-0.49	-0.41	-0.18	-0.48	-0.26	-1.38	-0.17	0.39	-0.71	-0.09	0.34	-0.81	0.00	0.02	-0.70	0.10	0.31

Figure 25: Chord diagrams of interactions between students segmented by ethnicity.



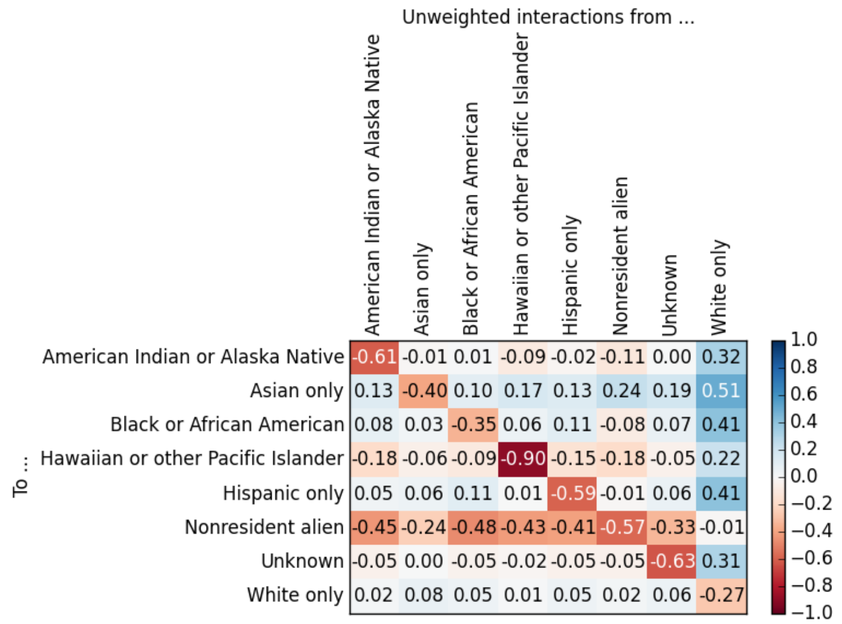
(a) Chord diagram of unweighted interactions between students segmented by ethnicity.



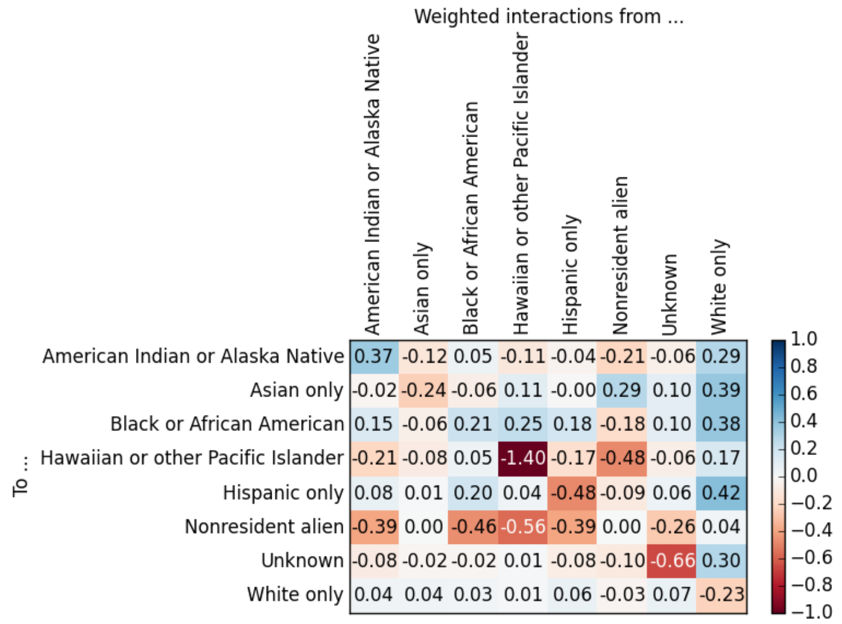
(b) Chord diagram of weighted interactions between students segmented by ethnicity.

Figure 26: RPD between actual and expected interactions of students segmented by ethnicity.

(a) RPD between actual and expected unweighted interactions of students segmented by ethnicity.



(b) RPD between actual and expected weighted interactions of students segmented by ethnicity.



trend for American Indian or Alaska Native students, though we should be careful about reading too much into this, as the sample size is much smaller ($n = 108$). This is also the case for the Hawaiian or other Pacific Islander segment, of which just 22 students fall into.

Also worth discussing is that White only students appear to come into contact with those of other ethnicities much more often than we would expect. This comes at the expense of interacting with other white students, which is less than expected.

A final interesting feature of the heatmaps in Figure 26 is that Nonresident Alien students seem to be a very isolated community. Most interactions with them from other segments are much less than we would proportionally expect. We are likely seeing the effects of the fact that Nonresident Alien students have fewer connections than other ethnicities (recall Figure 20a), and therefore few students of other segments are connected to them.

3.3.3 Students segmented by major

Finally, we examine students connected by major. Again, we'll present the chord diagram of interactions first.

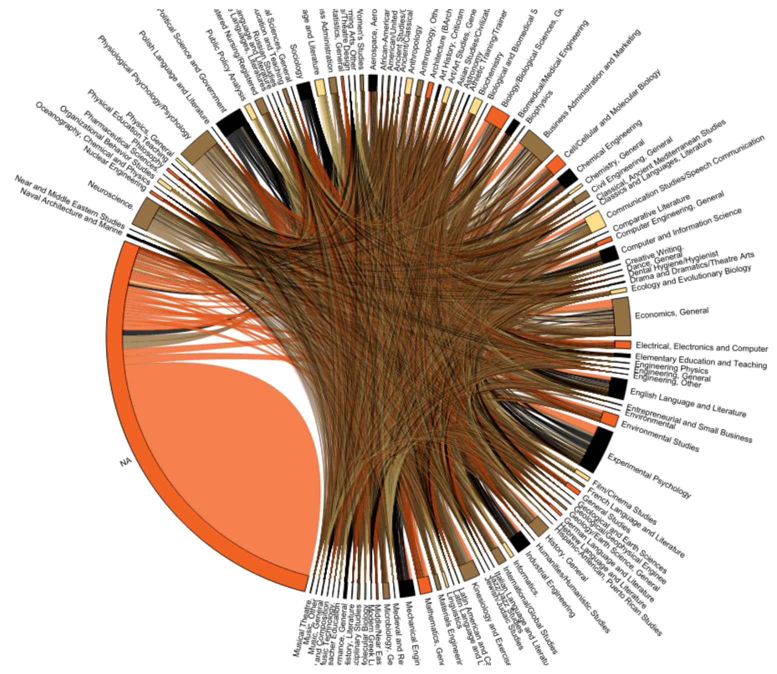
Unfortunately, we cannot display the heatmap here, it does not fit on the page. Instead, we go to a table again, like we did in Section 3.2.6. This time, however, let us study just the connections within the major. This will give us some interesting insight about how insular the major is.

A few features about Table 5 stand out. Firstly, the fact that there all the RPDs are positive. This makes sense: a student should be more connected in their major than we would expect for uniform interactions. Secondly, the majors that are more inwardly connected are those within specialized schools, especially music, theater, and performing arts majors. On the other hand, those majors at the bottom are some of the classical liberal arts majors, like Psychology, languages, and Math. The fact that majors like "General Studies" and "Multi-/Interdisciplinary Studies" appear there also makes sense, as these are the majors that encourage students to take a wider variety of classes, by definition. At the very bottom of both tables we see "NA". The fact that it is at the bottom is not surprising, it simply means that students without a declared major do not form as strong of a community as those students within their declared majors.

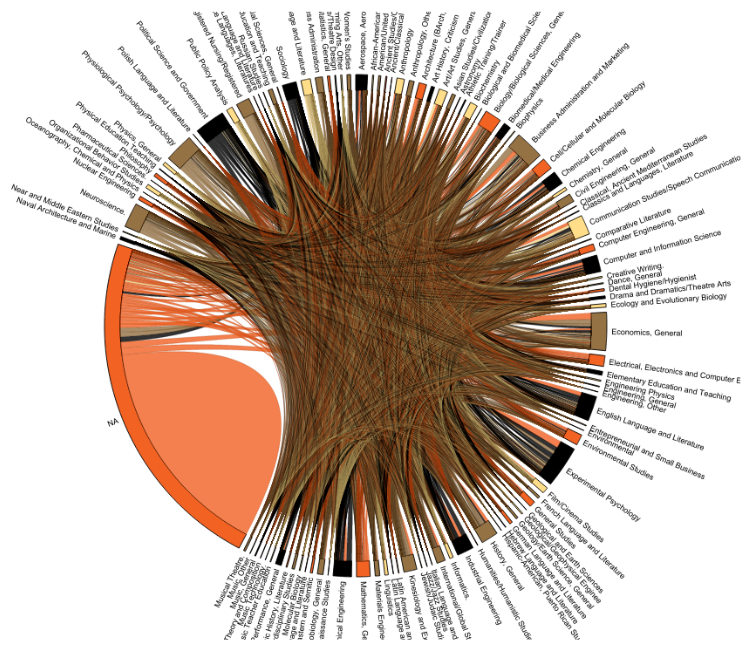
3.4 Examples of more and less connected students from various segments

Examination of which majors are more inconnected than we would expect them to be as we did in Section 3.3.3 leads us to the proposal of a new metric for richness of curriculum, which we investigate further in this section. This metric is simple: given a student v, it simply segments all other students u such that edge (u, v) exists by major, then measures the strength of v's "in"-connections vs. "out"-connections. "In"-connections are connections

Figure 27: Chord diagrams of interactions between students segmented by major.



(a) Chord diagram of unweighted interactions between students segmented by major.



(b) Chord diagram of weighted interactions between students segmented by major.

Table 5: RPD between actual and expected interactions of students segmented by major. We list the top 15 and bottom 15 majors.

(a) RPD between actual and expected unweighted interactions of students segmented by major

(b) RPD between actual and expected weighted interactions of students segmented by major.

Major	RPD	Major	RPD
Music Technology.	1.71	Music Theory and Composition	1.99
Musical Theatre.	1.69	Physical Education Teaching	1.99
Jazz/Jazz Studies	1.65	Dance, General	1.99
Technical Theatre/Theatre Design	1.63	Technical Theatre/Theatre Design	1.99
Art/Art Studies, General	1.63	Musical Theatre.	1.99
Music History, Literature	1.61	Athletic Training/Trainer	1.98
Music Theory and Composition	1.60	Music History, Literature	1.98
Music Performance, General	1.60	Dental Hygiene/Hygienist	1.98
Dance, General	1.60	Geological/Geophysical Enginee	1.98
Drama and Dramatics/Theatre Arts	1.49	Jazz/Jazz Studies	1.98
Visual and Performing Arts, Other	1.49	Music Technology.	1.98
Physical Education Teaching	1.48	Music, Other	1.97
Music Teacher Education	1.46	Geological and Earth Sciences	1.97
Hispanic-American, Puerto Rican Studies	1.44	Latin Language and Literature	1.97
Dental Hygiene/Hygienist	1.43	Music Teacher Education	1.97
Entrepreneurial and Small Business	0.76	Sociology	1.57
Mathematics, General	0.75	Spanish Language and Literature	1.57
Political Science and Government	0.75	Multi-/Interdisciplinary Studies	1.56
Biology/Biological Sciences, General	0.74	Entrepreneurial and Small Business	1.56
History, General	0.74	General Studies	1.50
Environmental Studies	0.74	English Language and Literature	1.49
Physiological Psychology/Psychology	0.73	History, General	1.47
Experimental Psychology	0.72	Engineering, General	1.44
Jewish/Judaic Studies	0.71	Neuroscience.	1.39
Economics, General	0.68	Biology/Biological Sciences, General	1.38
Pharmaceutical Sciences.	0.67	Physiological Psychology/Psychology	1.32
Spanish Language and Literature	0.66	Economics, General	1.31
Engineering, Other	0.59	Political Science and Government	1.28
General Studies	0.53	Experimental Psychology	1.22
Multi-/Interdisciplinary Studies	0.36	Engineering, Other	1.17
NA	0.24	NA	0.19

to students in the same segment as v (in this case, the same major), and "out"-connections are connections to any other segment.

We choose to investigate students in three specific majors: Musical Theatre, Philosophy, and General Studies. We have chosen these majors as representative of different samples on the in-connectivity spectrum as given by Table 5. Musical Theatre is a major that is very inwardly connected, General Studies has a much smaller in-connection strength, and Philosophy fits somwhere in between. These majors were also chosen for the sample size of students that we have in the data. All are on the order of magnitude of 100, not so small that we do not have a reasonable distribution, but small enough to make analysis possible to do locally, which is much more convenient.

We have chosen major as the way in which to segment because we believe that in-connections in it are more indicative of a students exposure to diverse ideas than in-connections in other segments. We make both a theoretical argument and an empirical appeal to support this claim. Theoretically, students' choice of major has a large impact on their collegiate experience. What they study will impact their future career and controls the ideas and concepts to which they are exposed. Empirically, in-connections between all majors are positive, indicating that these groups are more consistently communities than other segmentation schemes we have explored. Because we wish to measure how much students break out of their communities for exposure to different ideas, in- vs. out-connections has significance for this segment.

3.4.1 Musical Theatre students

It's a good idea to understand the distribution of in-connections before jumping into analyzing individual cases. Therefore, we present the distribution of Musical Theatre students RPD between actual and expected in-connections.

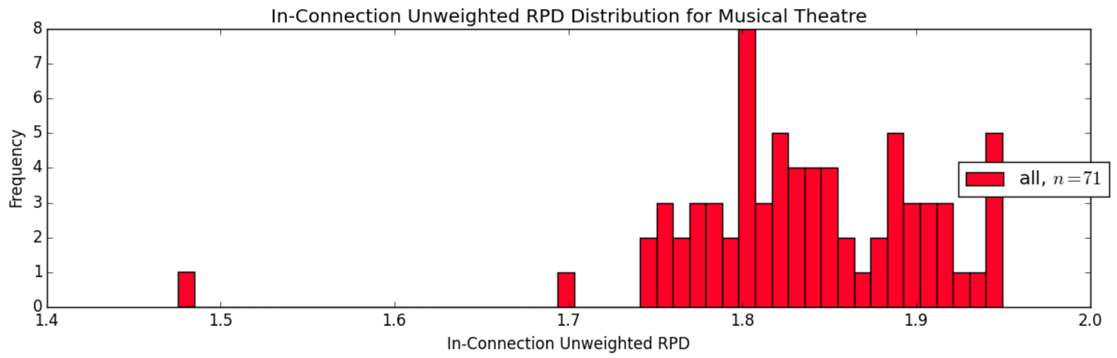
Based on the distributions in Figure 28, we may have to revise our plan a little on which students to study. Because the variance on these distributions is so small, a "typically" connected Musical Theatre student is one who is not very connected. Therefore, for this distribution, we will take just two case studies.

A more connected Musical Theatre student We'll start by examining the most connected Musical Theatre student, student 483249, who has unweighted in-connection RPD of 1.48 and weighted in-connection RPD of 1.94. Student 483249's classes are listed in Table 6

Despite the large number of classes student 483249, however, nearly all of them appear to be in music, with MUSTHTRE, THTREMUS, VOICE, PIANO, and THEORY making appearances. There is a streak of English classes mixed in, and indeed, student 483249 is a double major in English and Musical Theatre. Perhaps this is the reason he is one of the most connected Musical Theatre students. It's important to note, however, that even the

Figure 28: Distribution of RPD between actual and expected in-connections for Musical Theatre students.

(a) Distribution of RPD between actual and expected unweighted in-connections for Musical Theatre students. Median 1.83



(b) Distribution of RPD between actual and expected weighted in-connections for Musical Theatre students. Median 1.99.

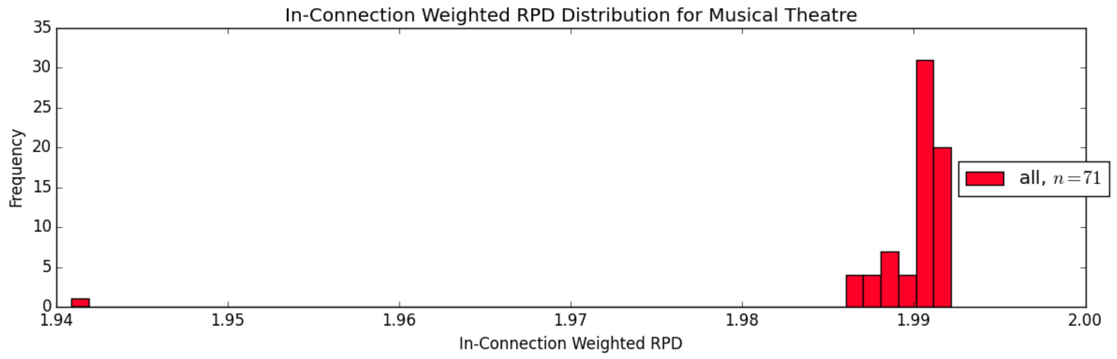


Table 6: A more connected Musical Theatre student's course schedule. Unweighted in-connection RPD: 1.48. Weighted in-connection RPD: 1.94.

BIOLOGY 171	BIOLOGY 172	BIOLOGY 173	DANCE 100
DANCE 100	DANCE 100	ENGLISH 125	ENGLISH 223
ENGLISH 297	ENGLISH 298	ENGLISH 315	ENGLISH 367
ENGLISH 398	ENGLISH 444	ENGLISH 450	MUSTHTRE 123
MUSTHTRE 123	MUSTHTRE 133	MUSTHTRE 134	MUSTHTRE 151
MUSTHTRE 152	MUSTHTRE 235	MUSTHTRE 236	MUSTHTRE 253
MUSTHTRE 254	MUSTHTRE 335	MUSTHTRE 351	MUSTHTRE 436
MUSTHTRE 441	MUSTHTRE 442	MUSTHTRE 480	MUSTHTRE 496
PHIL 232	PIANO 111	PIANO 112	PSYCH 111
SPANISH 232	STATS 350	THEORY 135	THEORY 236
THTREMUS 101	THTREMUS 102	THTREMUS 182	THTREMUS 240
THTREMUS 242	THTREMUS 245	THTREMUS 250	THTREMUS 251
THTREMUS 281	THTREMUS 282	THTREMUS 321	THTREMUS 323
THTREMUS 342	THTREMUS 382	THTREMUS 401	VOICE 100
VOICE 150	VOICE 150	VOICE 222	VOICE 423
VOICE 426			

most connected Musical Theatre student still takes a large number of classes in music, thus why on an absolute scale, he is not very well connected.

A less connected Musical Theatre student: If student 483249 is the most connected student, we can only imagine what a less connected student looks like. We consider student 186271, who has an unweighted in-connection RPD of 1.95 and a weighted in-connection RPD of 1.99. Table 7 can explain why.

These classes are even less diverse than our connected Musical Theatre student, as we expect. Beyond the four classes listed at the beginning and one "RCIDIV" (Residential College Interdivisional), not a single one of the courses Student 186271 took falls outside of Music. While this course schedule is bad for meeting new people at the University, it is good for our model because it shows that it can identify those who are less connected.

3.4.2 Philosophy students

The distribution of relative percent differences between actual and expected in-connections for Philosophy students is provided in Figure 29.

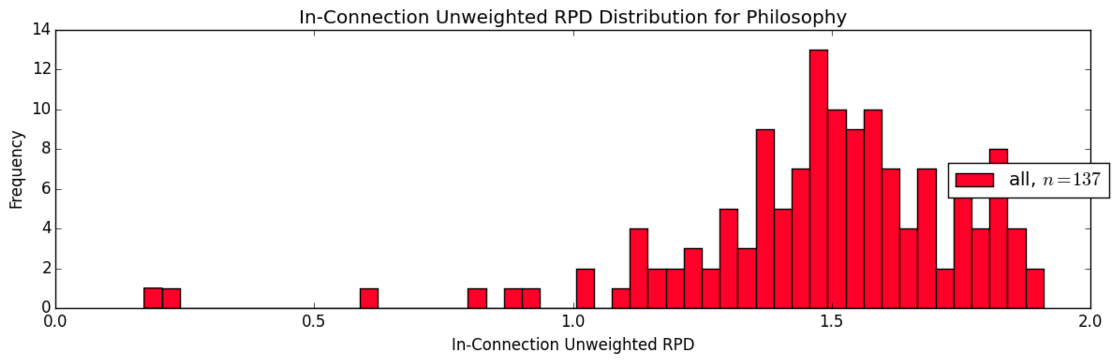
A more connected Philosophy student: We'll start by looking at student 222007, who has weighted in-connection RPD of 1.45 and unweighted in-connection RPD of 0.17, on the low end of both distributions. His course schedule is presented in Table 8.

Table 7: A less connected Musical Theatre student's course schedule. Unweighted in-connection RPD: 1.95. Weighted in-connection RPD: 1.99.

AMCULT 204	COMPLIT 122	ENGLISH 298	ENGLISH 467
MUSPERF 412	MUSTHTRE 123	MUSTHTRE 124	MUSTHTRE 133
MUSTHTRE 134	MUSTHTRE 151	MUSTHTRE 152	MUSTHTRE 235
MUSTHTRE 236	MUSTHTRE 253	MUSTHTRE 253	MUSTHTRE 253
MUSTHTRE 254	MUSTHTRE 280	MUSTHTRE 280	MUSTHTRE 280
MUSTHTRE 323	MUSTHTRE 324	MUSTHTRE 335	MUSTHTRE 351
MUSTHTRE 357	MUSTHTRE 407	MUSTHTRE 436	MUSTHTRE 441
MUSTHTRE 442	MUSTHTRE 450	MUSTHTRE 480	MUSTHTRE 480
MUSTHTRE 496	PIANO 111	PIANO 112	RCIDIV 302
THEORY 135	THEORY 236	THTREMUS 182	THTREMUS 211
THTREMUS 250	THTREMUS 281	THTREMUS 281	THTREMUS 282
THTREMUS 395	THTREMUS 399	THTREMUS 399	THTREMUS 435
VOICE 219	VOICE 220	VOICE 221	VOICE 222
VOICE 423	VOICE 425	VOICE 426	

Figure 29: Distribution of RPD between actual and expected in-connections for Philosophy students.

(a) Distribution of RPD between actual and expected unweighted in-connections for Philosophy students. Median 1.51



(b) Distribution of RPD between actual and expected weighted in-connections for Philosophy students. Median 1.91.

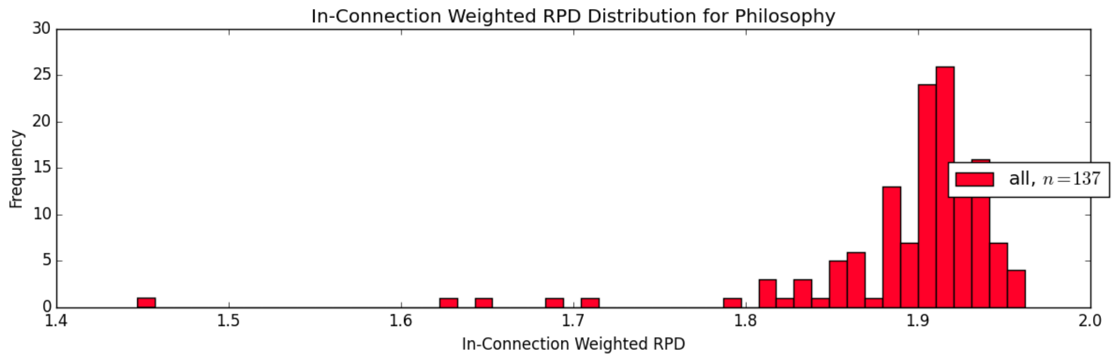


Table 8: A more connected Philosophy student's course schedule. Unweighted in-connection RPD: 0.17. Weighted in-connection RPD: 1.45.

AMCULT 100	AMCULT 201	ANTHRBIO 161	ARCH 218
BIOLOGY 171	BIOLOGY 172	CHEM 125	CHEM 126
CHEM 130	CHEM 210	CHEM 211	CHEM 215
CHEM 216	ECON 101	ECON 102	ECON 320
ENGLISH 125	ENGLISH 225	HISTORY 284	MATH 115
PHIL 232	PHIL 240	PHIL 345	PHIL 356
PHIL 366	PHIL 388	PHIL 389	PHIL 413
PHIL 420	PHIL 463	PSYCH 111	SOC 102
SPANISH 103	SPANISH 231	SPANISH 232	WRITING 410

Table 9: A typically connected Philosophy student's course schedule. Unweighted in-connection RPD: 1.46. Weighted in-connection RPD: 1.91.

ANTHRARC 284	ASTRO 127	CHEM 125	CHEM 126
CHEM 130	ECON 101	ENGLISH 124	GEOSCI 107
HISTORY 327	MATH 115	PHIL 202	PHIL 303
PHIL 359	PHIL 366	PHIL 383	PHIL 389
PHIL 402	PHIL 406	PHIL 485	POLSCI 111
POLSCI 140	POLSCI 314	POLSCI 320	POLSCI 327
POLSCI 389	POLSCI 432	POLSCI 489	PUBPOL 201
SPANISH 101	SPANISH 232	STATS 350	SWC 200

From Table 8, we see quite a few natural science courses in CHEM and BIOLOGY. Perhaps student 222007 planned on going to medical school before choosing a Philosophy major. Regardless, his courses seem to have connected him to that community. Additionally, there are a few one-off courses, such as Architecture and Writing in his transcript, which may have also given him exposure to other communities.

A typically connected Philosophy student: Student 279363 fits the bill of a typically connected Philosophy student. Her unweighted in-connection RPD is 1.46 and weighted in-connection RPD is 1.91. Her class schedule is given by Table 9.

Student 279363's classes show that in addition to Philosophy classes, she took quite a few politics classes: several POLISCI and a PUBPOL as well. In fact, student 279363 is a double major in Philosophy and Political Science, explaining the high concentrations of both courses we see. Beyond that, there do not appear to be many other courses that she took beyond those that appear to satisfy LSA distribution requirements. The two communities she is connected to by her double majors seem to be offset by the fact that she did not take very many classes in other disciplines.

Table 10: A less connected Philosophy student. Unweighted in-connection RPD: 1.96. Weighted in-connection RPD: 1.90.

ASIANLAN 101	ASIANLAN 102	ASIANLAN 201	ASIANLAN 202
GTBOOKS 291	PHIL 303	PHIL 345	PHIL 359
PHIL 361	PHIL 401	PHIL 409	PHIL 413
PHIL 420	PHIL 456	PHIL 460	PHIL 576
PHIL 585			

Table 11: A more connected General Studies student's course schedule. Unweighted in-connection RPD: -0.15. Weighted in-connection RPD: -0.47.

EEB 380	FRENCH 103	HISTORY 314	HISTORY 328
HISTORY 329	LING 272	MKT 311	MKT 313
MKT 325	POLSCI 318	PSYCH 230	PSYCH 260
PSYCH 280	PSYCH 345	PSYCH 436	PSYCH 448
PSYCH 476	SOC 461		

A less connected Philosophy student: Student 137985 is without a doubt a Philosophy student with few connections to other majors. In fact, he is the least connected Philosophy student in our sample in terms of both unweighted in-connection RPD and weighted in-connection RPD, which are 1.96 and 1.90 respectively. It becomes clear why this is the case when studying his course schedule in Table 10.

Student 137985 has taken only one class outside ASIANLAN and PHIL, a Great Books course. The fact that he is very unconnected is not surprising, his course schedule does not show very much diversity. Perhaps this is because he is a transfer student and managed to satisfy most of his distribution requirements through his transfer credit. Despite the fact that he is a transfer student, the fact that we can identify him as a non-connected student suggests that our model of examining in-connections does succeed to some extent.

3.4.3 General Studies students

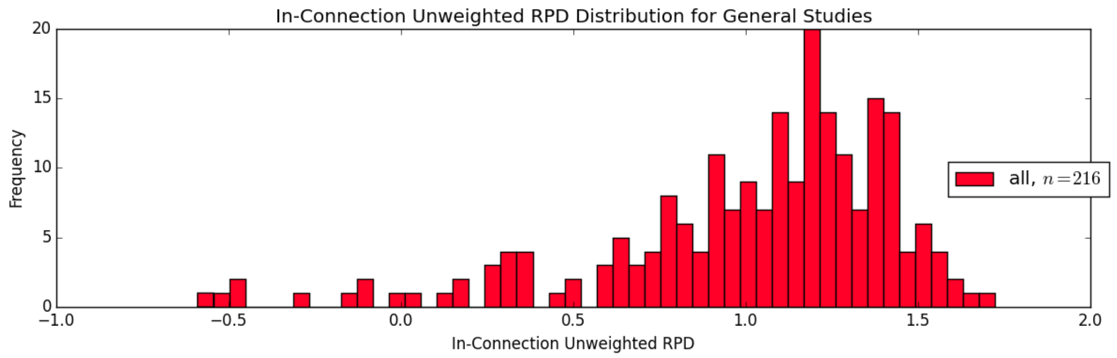
Again, for reference, we look at the distribution of relative percent differences between actual and expected in-connections for General Studies students. See Figure 30.

A more connected General Studies student: We'll start by considering a connected General Studies student, ID 398954. This male student has an weighted in-connection RPD of -0.47 and an unweighted in-connection RPD of -0.15, which means he is less connected to students in his own major than we would expect given a random graph. Let's consider his classes to see if we can make sense of this.

Indeed, upon inspection of Table 11, we can see that student 398954's schedule included courses from Psychology, Marketing, and History; perhaps he was studying the psychology

Figure 30: Distribution of RPD between actual and expected in-connections for General Studies students.

(a) Distribution of RPD between actual and expected unweighted in-connections for General Studies students. Median 1.12.



(b) Distribution of RPD between actual and expected weighted in-connections for General Studies students. Median 1.50.

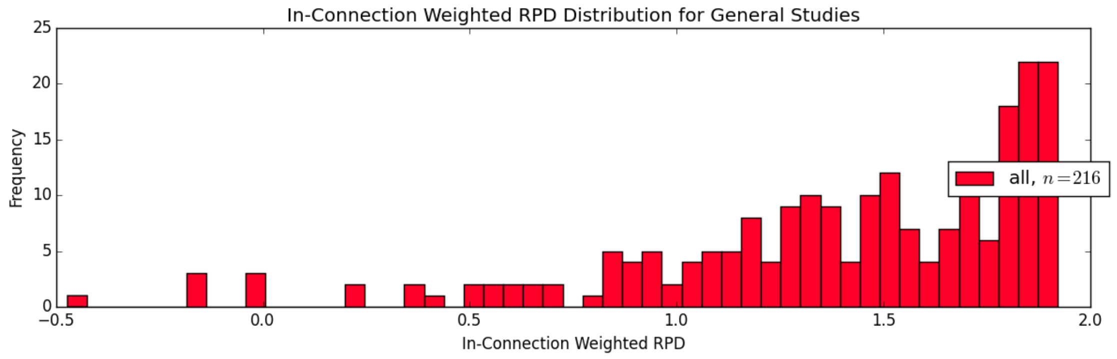


Table 12: A typically connected General Studies student's course schedule. Unweighted in-connection RPD: 1.21. Weighted in-connection RPD: 1.51.

AMCULT 301	AMCULT 421	ANTHRBIO 161	ANTHRCUL 330
ASIANLAN 125	ASIANLAN 151	COMPLIT 430	ENGLISH 125
ENGLISH 223	ENGLISH 239	ENGLISH 240	ENGLISH 313
ENGLISH 323	ENGLISH 340	ENGLISH 418	ENGLISH 484
LING 315	LING 316	LING 340	LING 370
LING 394	PHIL 383	RELIGION 481	SOC 210
WOMENSTD 330	WOMENSTD 348		

Table 13: A less connected General Studies student's course schedule. Unweighted in-connection RPD: 1.72. Weighted in-connection RPD: 1.90.

AMCULT 204	AMCULT 321	AMCULT 335	AMCULT 399
ANTHRCUL 411	ANTHRCUL 439	CAAS 111	CAAS 201
CAAS 202	CAAS 231	CAAS 333	CAAS 359
CAAS 360	CAAS 444	CAAS 450	CAAS 451
CAAS 495	CLCIV 372	EDUC 310	EDUC 310
EDUC 360	ENGLISH 125	HISTORY 303	HISTORY 347
JAZZ 454	LACS 305	LING 111	MOVESCI 110
PSYCH 200	PSYCH 218	PSYCH 327	PSYCH 353
SM 101	SM 111	SM 203	SOC 310

of marketing. His course schedule likely connected him not only to students in LSA, but also students in Ross.

A typically connected General Studies student: Student 385115 resembles a "typically" connected General Studies student, with weighted in-connection RPD of 1.51 and unweighted in-connection RPD of 1.21, both right around the median of the distribution.

Based on Table 12 Student 385115 appears to be a little more focused on the language track than Student 398954 showed, with interest in culture. This more focused track along the soft humanities appears to have exposed him to more General Studies students.

A less connected General Studies student: Student 283971 is certainly one of the least connected General Studies students, with an unweighted in-connection RPD of 1.72, and a weighted in-connection RPD of 1.90. Student 283971's courses are listed in Table 13.

Student 283971 has an interesting pattern of classes, focusing mostly on cultural topics, but with some dabbling in Kinesiology classes like Movement Science "MOVESCI" and Sports Management "SM". One theory is that this student is an athlete. Indeed, it appears that many more Athletes proportionally are General Studies majors than we would expect based on their numbers on campus. One article quotes that 49% of General Studies majors are athletes, while just 3% of students on campus are athletes [14]. If indeed student 283971 is an athlete, he likely took classes with other athletes, thus giving him a very high in-connection RPD.

4 Results and Conclusion

Before we discuss the larger ramifications of the analysis of our data, it's useful to remember what our initial goal was: to characterize the intellectual connections that a student has over the time they spend at a University. We want to be able to quantify the diversity of ideas a student comes into contact with and through this, measure the richness of their undergraduate experience. We take this as axiomatically important for the liberal arts education that a university seeks to provide.

We admit that the study of degree distributions in Section 3.2 is not particularly illuminating. It confirms much of what we expect: that students with fewer years at the University have fewer intellectual connections than those who have spent more years. It also lets us identify particular groups of students who lack the number of connections that their peers have, like transfer students and Nonresident Alien students.

Similarly, the analysis in Section 3.3 raises some interesting questions, but not many answers. Particularly interesting are the questions raised by the ethnicity interaction heatmap in Figure 26. Each one of these cells could be analyzed from a sociological perspective to try and make sense of actual number of interactions vs. expected number of interactions, but this analysis is beyond the scope of this thesis.

Section 3.3 does, however, lead us to the interesting realization of how important communities of students with the same major are. The data clearly show that students interact with others in their major much more than they would at random, giving empirical basis to the idea that majors are a useful community, something that we would expect to be true. Because majors are a meaningful community, looking at the relative percent difference of actual in-connections among students of a particular major and the expected in-connections given a random graph actually has meaning. We can see which students take classes more confined to their community and which branch out to take different classes with other students.

When we looked at in-connectedness across majors we see that it varies quite a bit. For example, Musical Theatre students are very in-connected; the most out-connected student had a weighted RPD of 1.94, higher than the least-connected General Studies student, who had a weighted relative percent difference of 1.90. When we studied the course schedules of these students, we saw that their course schedules often made sense of their in-connection RPD. Musical Theatre students take an overwhelming amount of musical theater classes, connecting them to a very small community: other Musical Theatre students, while General Studies students take classes across the University, even when they have a relatively high in-connection RPD.

With this in mind, we propose in-connection RPD as a heuristic to measure a student's richness of curriculum and exposure to intellectual diversity. Specifically, the lower the in-connection RPD, the more exposure a student has to other ideas in the University. We have discussed that majors as a community make sense both theoretically and empirically, and by measuring in-connection RPD, we get a sense of just how connected a student is to that community. Students who have fewer in-connections are the students who are willing to take classes outside their comfort zone, to expose themselves to different view points that they have not seen before.

That said, we'd like to emphasize the preliminary nature of this work. While we have shown that in-connection RPD is a useful heuristic, it by no means is the only way to measure a student's intellectual exposure. In the following sections, we discuss other interesting analyses that can be done with the data.

4.1 Further analysis work

Our measure of major in-connection only scratches the surface of graph measures that we can use on G_s. Specifically, while we identified majors as an externally defined community that is internally quantifiable and has utility when considering student connections, other communities likely exist as well. Michigan Learning Communities, such as the Honors Program, are not explicitly defined in the data set that we are given, but likely have some practical significance in determining which classes students take. By requiring incoming freshmen to take Great Books, for example, the Honors Program forges intellectual connections among its students. If we can identify communities like this in the data, we can also measure in-connection RPD of them as well.

We also might be able to find stronger communities, where in-connection RPD would be stronger than that for just majors. Examining these groups would be especially interesting for our analysis, because students with a smaller in-connection RPD here would show a willingness to take classes without their tight-knit community. By taking a class without the protective bubble of friendship, we'd expect students to feel more exposed to the ideas expressed in the class, which is exactly what we're trying to measure.

Beyond in-connections in communities, we also might wish to look at certain interesting individuals. One way we can quantify this is with betweenness. Specifically, how many shortest paths between students in G_s go through a specific person? This measure likely corresponds to a more intellectually connected individual who has taken classes in various disciplines. We might call people with high betweenness intellectual "connectors".

Let us also not forget the course network G_c, where edges between courses are weighted according to the number of people who enrolled in each course. We could perhaps use this to identify courses which connect students unlikely to be connected without them. Also, while somewhat orthogonal to the goal of characterizing which students take courses that connect them to different students, analyzing communities of courses could be a useful took for students who are looking to expand their intellectual breadth.

4.2 Further technical work

Due to the scale of the network, some additional technical work needs to be done to facilitate this analysis. In particular, betweenness finding is incredibly algorithmically intensive. The Boost Graph library has a betweenness centrality algorithm which gives a time complexity of $O(n^3 \log n)$ for weighted graphs and $O(n^3)$ for unweighted graphs. Even with the backing of Flux, the runtime cost of finding the betweenness centrality for every student in G_s would likely be far beyond feasible.

For this reason, a method for taking random samples of the graph is likely needed to be able to scale the problem to a more reasonable size. How to do this is not immediately evident. In their paper on sampling from large graphs, Jure Leskovec and Christos Faloustos present random node sampling, random edge sampling, and random walk sampling among other sampling methods [18]. They use several realism criteria to evaluate these methods of sampling. To perform graph sampling on G_s, we would have to evaluate which of these criteria are useful and also have to implement the sampling method.

Less difficult but still nontrivial to achieve is a cleaner data set. Our data set has the unfortunate property of having students without all their connections. We can see this particularly in Section 3.2.1, where students who have been in school for just a few years lack the number or strength of connections of upperclassmen. Another issue that cleaning up the dataset would solve is the fact that years at the edges of the data set lose many of their connections. Ideally, we would analyze a few clean years, say 2008 to 2010, to reduce noise. This is not quite as simple as cutting out students from years outside our "clean range", because students who entered in 2008 to 2010 are still connected to students on the edges.

4.3 Final thoughts

For many in college, GPA has overwhelming importance. For those planning on continuing their academic career, GPA can be a barrier from receiving admission into various schools. Even employers might use GPA to screen candidates. While it would be disingenuous to claim that GPA is "just a number", GPA does reduce the undergraduate experience to just one measure: performance in classes. The overwhelming focus on this number can lead to loss of focus on other aspects of the collegiate experience, like exposure to new and different ideas, the hallmark of a liberal-arts education.

While some might claim that exposure to diverse ideas and perspectives is a futile quest when, ultimately, gaining employment depends on skill sets, this idea is misguided. Within the last year, many of the giants in the software industry, my industry of interest, have issued diversity reports giving some demographic information about the company [10]. The attention these reports have received show that working with a diverse group of peers is important.

Despite this, little research has been done to quantify students' exposure to different perspectives while they are undergraduates, arguably the time in which they grow the most

intellectually. This thesis presents preliminary work looking to quantify students' connections with others. Ultimately, the goal of this work would be to produce a number that can serve alongside the GPA to give an idea of a students' intellectual breadth, thereby incentivizing not just academic performance, but challenging their beliefs.

This research was supported in part through computational resources and services provided by Advanced Research Computing at the University of Michigan, Ann Arbor.

References

[1] Electrical engineering & computer science undergraduate workload survey (spring 2014). Accessed: 2015-04-08.

[2] Lars Backstrom, Dan Huttenlocher, Jon Kleinberg, and Xiangyang Lan. Group formation in large social networks: Membership, growth, and evolution. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 44–54, New York, NY, USA, 2006. ACM.

[3] Jean A Baker. Teacher-student interaction in urban at-risk classrooms: Differential behavior, relationship quality, and student satisfaction with school. *The Elementary School Journal*, pages 57–70, 1999.

[4] A.L Barabsi, H Jeong, Z Nda, E Ravasz, A Schubert, and T Vicsek. Evolution of the social network of scientific collaborations. *Physica A: Statistical Mechanics and its Applications*, 311(34):590 – 614, 2002.

[5] H.Russell Bernard, Eugene C. Johnsen, Peter D. Killworth, Christopher McCarty, Gene A. Shelley, and Scott Robinson. Comparing four different methods for measuring personal social networks. *Social Networks*, 12(3):179 – 215, 1990.

[6] Christy M.K. Cheung, Pui-Yee Chiu, and Matthew K.O. Lee. Online social networks: Why do students use facebook? *Computers in Human Behavior*, 27(4):1337 – 1343, 2011. Social and Humanistic Computing for the Knowledge Society.

[7] Berman Daws. Boost background information, 2005. Accessed: 2015-04-08.

[8] Patrick Doreian. On the connectivity of social networks. *The Journal of Mathematical Sociology*, 3(2):245–258, 1974.

[9] Nicole B. Ellison, Charles Steinfield, and Cliff Lampe. The benefits of facebook friends: social capital and college students use of online social network sites. *Journal of Computer-Mediated Communication*, 12(4):1143–1168, 2007.

[10] Conner Forrest. Diversity stats: 10 tech companies that have come clean, 2014. Accessed: 2015-04-08.

[11] Linton C. Freeman. Centrality in social networks conceptual clarification. *Social Networks*, 1(3):215 – 239, 19781979.

[12] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826, 2002.

[13] Caroline Haythornthwaite. Social networks and internet connectivity effects. *Information, Communication & Society*, 8(2):125–147, 2005.

[14] John Heuser, Dave Gershman, and Jim Carty. University of michigan athletes' 'safe harbor' is general studies, 2008. Accessed: 2015-04-08.

[15] Norman P. Hummon and Patrick Dereian. Connectivity in a citation network: The development of dna theory. *Social Networks*, 11(1):39 – 63, 1989.

[16] Jens Krause, David Lusseau, and Richard James. Animal social networks: an introduction. *Behavioral Ecology and Sociobiology*, 63(7):967–973, 2009.

[17] Ravi Kumar, Jasmine Novak, and Andrew Tomkins. Structure and evolution of online social networks. In Philip S. Yu, Jiawei Han, and Christos Faloutsos, editors, *Link Mining: Models, Algorithms, and Applications*, pages 337–357. Springer New York, 2010.

[18] Jure Leskovec and Christos Faloutsos. Sampling from large graphs. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 631–636, New York, NY, USA, 2006. ACM.

[19] David Lusseau. The emergent properties of a dolphin social network. *Proceedings of the Royal Society of London. Series B: Biological Sciences*, 270(Suppl 2):S186–S188, 2003.

[20] David Lusseau and M. E. J. Newman. Identifying the role that animals play in their social networks. *Proceedings of the Royal Society of London. Series B: Biological Sciences*, 271(Suppl 6):S477–S481, 2004.

[21] Riccardo Marcangelo. N4017 - non-member size() and more, 2014. Accessed: 2015-04-08.

[22] Daniel Muijs, Mel West, and Mel Ainscow. Why network? theoretical perspectives on networking. *School Effectiveness and School Improvement*, 21(1):5–26, 2010.

[23] M. E. J Newman. *Networks: An Introduction.* Oxford University Press, 2010.

[24] Zizi Papacharissi. The virtual geographies of social networks: a comparative analysis of facebook, linkedin and asmallworld. *New Media & Society*, 11(1-2):199–220, 2009.

[25] Colin Pilbeam, Gaynor Lloyd-Jones, and David Denyer. Leveraging value in doctoral student networks through social capital. *Studies in Higher Education*, 38(10):1472–1489, 2013.

[26] T Roger and David W Johnson. Cooperative learning, 1994.

[27] Herb Sutter. Gotw #32: Preprocessor macros, 2009. Accessed: 2015-04-08.

[28] Herb Sutter. Elements of modern c++ style, 2011. Accessed: 2015-04-08.

[29] Herb Sutter. N4165 - unified call syntax, 2014. Accessed: 2015-04-08.

[30] Nashrawan Taha and Andrew Cox. International students' networks: a case study in a uk university. *Studies in Higher Education*, 0(0):1–17, 0.

[31] Beverly Daniel Tatum. *"Why are all the Black kids sitting together in the cafeteria?": and other conversations about race.* Basic Books, 2003.

[32] Graduation honors. Accessed: 2015-04-08.

[33] Enrollment reports. Accessed: 2015-04-08.