# Good Bull Schedules

Informing Students' Course Scheduling Decisions

Gabriel Britain, Suvedh Srikanth

## 1 Introduction

In this report, we introduce an open-source, open-data scheduling application. The application boasts a high-performance search endpoint, allowing for students to quickly search for courses based not only on name and course number, but also on course description. In addition, the application enables students to graphically plan out multiple schedules at once, while at a glance viewing historical grade distributions.

## 2 Background and Motivation

In Fall 2019, Texas A&M University will offer 17,934 sections in 4,278 distinct courses for students. This number is increasing dramatically every year, and yet no official method for planning schedules is offered by the university. To plan for the upcoming semester, students have to cross-reference multiple data sources (discussed in the Web Crawling section), and then graphically chart out availability. The goal of this project is to unite these data sources into an intuitive application for students to plan their schedules with ease, so students can be well-informed on the courses they plan to take. In addition, we hope to construct a platform for alternative uses of this data.

## 3 Previous Approaches

A number of different approaches to planning out schedules have emerged. The foremost approaches at the time of writing include manual methods, such as charting availability using pencil and paper, using spreadsheets, and even using third-party commercial websites. None of these approaches are both efficient and tailored specifically to Texas A&M University students.

## 4 Technical Explanation

Since there are no public official APIs for course data, our data comes exclusively from crawling and parsing HTML or PDFs. The contributions of this project

can be broken into three separate categories, each of which presented significant technical challenges.

## 4.1 Web Crawling

One of the most difficult parts of creating our application was scraping three different sources of data, offered in wildly different formats and varying levels of machine readability. Sections offered per term are displayed on Compass, while course descriptions are displayed on the course catalog. Grade distributions are only offered in PDF format, which makes them difficult to study and analyze. Each data source must be crawled individually using different scrapers to collect important information.

When we first wrote the scrapers, each page (and PDF) was retrieved synchronously, meaning that requests were executed one after another, with a lot of hanging time between request and response. This made scraping each data source take a significant amount of time (approximately 10 minutes for courses, 7-8 hours for all sections dating back to 2009, and 30 minutes for grade distributions), which was far too slow. To claim that our data is up-to-date, we need to completely refresh our data source multiple times per day, and a latency of 7-8 hours was far too high.

We noticed some subproblems that could be executed independently, and came to the conclusion that we could complete execution much quicker if we moved to an asynchronous request model. We designed an asynchronous request pool solution using multiple threads that dramatically improved data collection times (approximately 30 seconds for courses, 1 hour for sections, and 10 minutes for grade distributions). This allows us to refresh our data source more often, and to assure our users that the data they are using to plan their upcoming semester is accurate.

More detail is given in the following sections regarding techniques used to crawl and parse data.

### 4.1.1 Courses

The Texas A&M University course catalog is arguably the least-painful of our data sources to scrape. The HTML is well-structured and predictable, and using Python's BeautifulSoup4 library, selecting elements from the HTML is simple. For tasks that require more complexity, data is extracted using regular expressions.

### 4.1.2 Grades

Programmatically retrieving grade distributions is more difficult than retrieving courses. Though the PDFs offered by the Office of the Registrar appear tabular, the data extracted from them by the library we use is not. After studying the extracted data carefully, we can discern patterns that marked the beginnings

and ends of rows. We constructed a rule-based parser that captures all of the necessary information from the PDFs in a timely manner.

#### 4.1.3 Sections

Retrieving sections presents the largest challenge. Unlike the previous two sources, Compass expects specific form data or query parameters to be sent alongside requests, which changes with each department and term. In order to simulate an actual user, we studied each network request made while retrieving sections, and then replicated the requests. From there, we face another challenge: processing poorly-structured HTML. To adequately extract the data we want, we pre-process the HTML using BeautifulSoup4, and then extract most of our data using regular expressions. Finally, we have to deal with strange, unpredictable server behaviors (such as random connection resets and un-prompted server errors), which we mitigate by simply re-sending our request.

However, in order to be respectful of server load, we also implemented a shallow-scraping mode that only scrapes recent terms. A full scraping procedure only has to run once, while the shallower procedure can be run as often as once an hour with no difficulty.

### 4.2 Indexing

As crawling is performed, our data is stored in different PostgreSQL tables that represent the relationships between courses, sections, instructors, and grades. However, simply storing and relating our data is not enough to build an intuitive scheduling application. We must provide our users with an interface to this data, and for that, we used Elasticsearch.

Initially, we indexed our data using Solr. However, as time went on, we found Solr to be rather inflexible, and most libraries to interface with it from Python to be deprecated. We switched to Elasticsearch, which offers a more flexible search functionality, and enables us to quickly and easily change our search schema.

We decided to use our scraped course data for search, as it provides the most verbose information. We want users to be able to search for courses not just on their title and course number, but on the material covered by the course, so topic-based searches become viable. After studying our data, we realized that many course descriptions reference topics by title, such as "Information Retrieval", or "Machine Learning", or "Data Structures", and the majority of those titles are 2-3 tokens in length. In addition, we noted that many of our users' searches were short queries of an approximately equal length. To better fit user searches, we opted to build indexes that captured token-based $n$-grams, otherwise known as $w$-shingles. Of course, to augment these shingled indexes, we construct TF-IDF indexes to further improve weighting.

By taking the extra time to study our data, we were able to reduce search query timing to under 100 milliseconds. In the future, if other manners of searching and filtering are desired, such as for courses offered in specific departments,

updating the indexes in Elasticsearch is simple.

## 4.3 Application Design

### 4.3.1 Backend

To construct a reliable backend webserver for our application, we elected to use Django. Django provides a number of commonly-used pieces of functionality (such as a flexible object-relational mapping, and user authentication) that enabled us to create extensible models and greatly simplify our codebase. There is also an excellent library for Elasticsearch that enables us to automatically index our course data using the Django ORM, which greatly simplifies the task of refreshing our search index.

One of the first things that we noticed after constructing our API was that requests for large courses (that offered hundreds of sections during a term) were taking too long (approximately 333 milliseconds). To improve query performance, we use Redis, a high-performance cache, which we configured to cache requested courses for an hour. This greatly improves our server's performance, and improves users' experience.

With all of these different applications interfacing with one another, we needed a way to provide consistency in our development environments. We use Docker to containerize our application, which not only simplifies setting up a development environment, but also simplifies the deployment process.

### 4.3.2 Frontend

Our front-end application is built using TypeScript, a superset of JavaScript that provides typechecking. To create an interactive user interface, we chose to use React, a popular front-end framework created by Facebook. Rather than taking the time to style each individual component ourselves, we used Google's Material Design Components for React to create a prototype of our scheduling application that is visually appealing. Our React application interacts with our Django backend by making requests to different endpoints. Users are authenticated using sessions, which ensures that they can leave the page and pick up where they left off in the future.

# 5 Statistics

Figure 1 simply displays the number of rows stored in the PostgreSQL database. Elasticsearch indexes all of the courses, both on course name and on course description, providing high-quality search results. Figure 2 displays the number of sections offered per campus per year. An interesting thing to note is the sharp spike in growth between 2018 and 2019's number of offered sections in College Station.

| Type | Count |
|---|---|
| Course | 11266 |
| Section | 305887 |
| Grades | 95746 |

Figure 1: Number of Rows

| Year | College Station | Galveston | Qatar |
|---|---|---|---|
| 2010 | 26445 | 1108 | 301 |
| 2011 | 23464 | 1204 | 331 |
| 2012 | 23443 | 1333 | 382 |
| 2013 | 23533 | 1292 | 407 |
| 2014 | 25117 | 1296 | 386 |
| 2015 | 26707 | 1503 | 377 |
| 2016 | 27647 | 1349 | 367 |
| 2017 | 28272 | 1358 | 369 |
| 2018 | 28445 | 1333 | 364 |
| 2019 | 41200 | 1277 | 373 |

Figure 2: Yearly Sections Per Campus

# 6 Future Work

In the future, we hope to incorporate a number of intelligent systems into the application. Firstly, we'd like to incorporate a course recommendation system, that bases its recommendations on students' course choices for the upcoming semester. In addition, we'd like to incorporate a student review system, similar to PICA evaluations, where students would leave their qualitative feedback on instructors. Sentiment analysis and topic modeling could both be used in this capacity to provide simple tags and classifications for instructors for students. Lastly, we'd like to employ the use of constraint solvers to enable students to provide a set of constraints (such as start and end time) and a set of courses, and have the solver generate a relatively optimal schedule.

# 7 Conclusion

In this report, we presented an open-source, open-data scheduling application that allows students to approach planning for the upcoming semester using concrete, quantitative data visualization. We hope that by open-sourcing this tool, we can encourage students to build on our platform and become better-informed about their university and quality of education.