Abstract

In today's economy, the efficient use of time and allocation of resources is invaluable. For many supervisors, these goals are achieved by assigning employees to scheduled shifts based on their availability and preferences. Generally, manual scheduling solutions can range from prioritizing task allocations on a first-come, first-serve basis to trial-and-error assignments. These methods are prone to both uneven distribution of shifts and even incomplete assignments. Manual scheduling can also quickly become a labor-intensive process that increases rapidly in difficulty with a larger number of employees.

I present HelpDesk, an application that addresses these issues and creates a scalable system to optimally accommodate scheduling needs. The system allows employees to specify which hours of the week they are available or would prefer to work. Once a manager or supervisor specifies all of the scheduled shifts, including time, duration, and even number of employees, HelpDesk produces a mathematically optimal assignment of employees to shifts. This process can be repeated for different input variables, which allows users to craft specific allocations to meet the needs of the employer.
**Problem**

On-campus work study jobs employ workforces of dozens of students. This means creating a schedule that meets the availabilities of every student. For CIS 110, a course with over 50 TAs and Mentors, scheduling which employees are assigned to each hour of office hours is a complicated, time-consuming problem. The current solution to this problem is to have each employee fill out their availabilities and preferences in a large spreadsheet. Once entirely filled out, the head TAs have to manually produce a schedule that both completely satisfies every opening in office hours as well as optimizing for employees’ preferences. This process has many inherent problems, the first of which being that scheduling cannot begin until all TAs have entered their availabilities. Once it has begun, it often takes several days in order to produce a schedule for every opening, and normally any possibility of optimization for preference is disregarded if a completed schedule cannot be made.

Another tangential problem for CIS 110 is the management of office hours. Several semesters ago, office hours for CIS 110 was handled like it is handled for any other class: names are written on a whiteboard and wiped off one-by-one as they are helped. As CIS 110 hit new maximums for enrollment, the whiteboard often filled the entire board with several columns of names. This was eventually replaced with an electronic system that was simply a proof-of-concept and only provides basic functionality. There is no way to query this system for information about metrics such as average case time or most popular office hours.

**Approach**

The system was designed firstly as a REST API, produced by a back-end server and consumed by a web-facing front-end interface. This was chosen for extensibility, both in features and possible front-end implementations. Ideally someone could develop mobile platform applications for HelpDesk that consume the same API. The back-end server is written in Java on top of the Dropwizard framework because it integrates many existing standard Java web technologies. The core component that takes care of the REST API is Jersey which is the reference implementation for the Java API for RESTful Services (JAX-RS). Since one of the

---

1 dropwizard.io/0.9.2/docs
2 jersey.java.net
3 jax-rs-spec.java.net
goals of the server design was extensibility on the API consumption side, every transfer was serialized and deserialized as JSON using the library Jackson⁴, because it has become the lingua franca for web development. The database connection is handled by the JDBI⁵ SQL library, which can connect to PostgreSQL, MySQL, MSSQL, etc. but used an H2⁶ in-memory database for development.

The initial step was developing the API, which then required designing the data structures (users, semesters, shifts, etc.) used by the system. Most of the functionality implemented before the scheduling optimization feature was completed was composed of basic create, modify, and delete commands on the core data structures of the system. The main scheduling functionality was implemented by structuring the data as an Integer Linear Programming problem. The objective function is set to maximize the number of preferred hours assigned to employees. The manager can also set constraints such as number of shifts per user (individually or for all users) or number of users per shift. Then this problem is passed to an ILP solver through the JavaILP⁷ wrapper library. The reference solver that is bundled with HelpDesk is SAT4J⁸, a free, pure-Java SAT solver. However, the JavaILP wrapper can be used in combination with other, more powerful solvers that require licenses such as the Gurobi Optimizer⁹ or IBM’s ILOG CPLEX¹⁰.

The front-end interface was developed using Angular 2¹¹ because the existing CIS 110 websites were developed using Angular and therefore there was a lot of relevant code to pull from when designing the system. However, it is worth noting that while they are structurally similar, Angular and Angular 2 are completely incompatible. As well, Angular 2 takes advantage of Microsoft’s TypeScript¹², a strict superset of the JavaScript language that introduces static typing. For connectivity between the front-end and the back-end, a custom REST adapter generator was built that automatically parses the JAX-RS annotations of the server and builds the appropriate TypeScript definitions for use on the front-end. This means that any changes to the server-side API can be quickly propagated to the front-end for ease of development.

---

⁴ wiki.fasterxml.com/JacksonHome  
⁵ jdbi.org  
⁶ h2database.com  
⁷ javailp.sourceforge.net  
⁸ sat4j.org  
⁹ gurobi.com/products/gurobi-optimizer  
¹⁰ ibm.com/software/commerce/optimization/cplex-optimizer/index.html  
¹¹ angular.io  
¹² typescriptlang.org
At this point, many of the existing server functions were given user interfaces. As well, different interfaces were constructed for different types of users (students, employees, managers, etc.) all on the same system. While it is automatically served up with the HelpDesk server on launch, the front-end could be deployed to a separate location and hosted statically, in order to decrease the resource footprint of the system.

Results

The system ended up working very well, especially for the use case of CIS 110. The previous semesters’ TA/Mentor preference data was compared against the final office hour schedule. The metric used to compare the effectiveness of these processes in this instance is average preference level. The HelpDesk system allows users to specify if they are unavailable, available or available and would prefer to work a given time slot. These values are weighted 0, 1, and 2 accordingly. This means that if a user had an average preference level of 1.5, that would indicate that they received 50% preferred shifts. The results from the previous two semesters of CIS 110 showed that the average preference levels were $1.773^{13}$ and $1.735^{14}$. When processed by HelpDesk, the same input values produced schedules with average preference levels of 1.931 and 1.930 respectively.

Ethical Considerations

This system does not really deal with many privacy concerns as all data is hidden from users that do not have the appropriate access levels. This feature was built-in to the system by design as it had to function within an academic environment where personal information and access to it has to be well safe-guarded. As for ethical concerns, this system does solve the problem of inherent bias in scheduling process. It may be that certain TAs that are more favored by the Head TAs are more likely to get their preferred hours. Or worse, someone disliked by the management may get their least sought-after working times. With this system, the human element is removed from the scheduling process, limiting the possibility of introducing any bias or prejudice.

---

$^{13}$ Fall 2015
$^{14}$ Spring 2016
Discussion

This system, while I believe it is strong in concept and will ultimately be so when put to use, is not without some limitations. The system is designed in such a way that everyone using the system is trustworthy. There is nothing stopping someone from only listing their preferred times, effectively gaming the system (which needs a large pool of availability to make a good schedule). As well, there are many features that are implemented on the server-side that never had front-end implementations made. This was mostly due to timing, but also that I have limited experience with front-end web development.