AggieAssign: An End-to-End Schedule Generator

Colby Endres
Department of Computer Science
Texas A&M University

***Abstract*—Scheduling is a classic problem in computer science and presents unique challenges in optimizing resources and satisfying constraints. We focus on generating a course schedule for a given semester's class offerings. This requires balancing a variety of factors, including room availability, class sizes, and professor aptitude. We present AGGIEASSIGN, an end-to-end application that creates customizable course schedules that respect instructor preferences. Using linear programming, AGGIEASSIGN can generate a schedule in under a minute- a significant improvement over manual techniques.**

I. MOTIVATION

AGGIEASSIGN was undertaken as a final project for CSCE 606. Our client was Dr. Bettati, one of the associate heads of the computer science department. One of his responsibilities is the formation of the department's semester course schedule. If done manually, this task requires a great deal of time and effort. Scheduling demands balancing physical constraints, such as room occupancy or extra classroom features necessary for lab courses. There are also soft preferences to consider, namely the instructor's affinity for course material and time slot. The core goal of this project was to algorithmically generate quality schedules, which could then be tweaked by hand if needed.

II. USER EXPERIENCE

We give a brief overview of how a potential user might use AGGIEASSIGN. After authenticating with a TAMU-associated email, the user can create a schedule environment. Each environment is isolated from other schedules, giving the client flexibility in generating many different schedules at once.

A. Data Requirements

Due to the nature of the problem, our algorithm requires a significant quantity of data from the user. The following are necessary components for scheduling:

- Courses to be offered in the semester
- Rooms allocated to the department for lectures
- Available professors
- Time slots for courses to be held

To be more specific, each class has a course ID, section ID, and a max enrollment. This allows us to differentiate between courses that are split into multiple sections. Rooms require a label, maximum capacity, and a value to mark if it is a learning studio, a classroom equipped with extra technology required by some lab courses. Professors must have a unique ID and a course load representing the maximum amount of contracted hours they can teach. Additionally, we provide support for

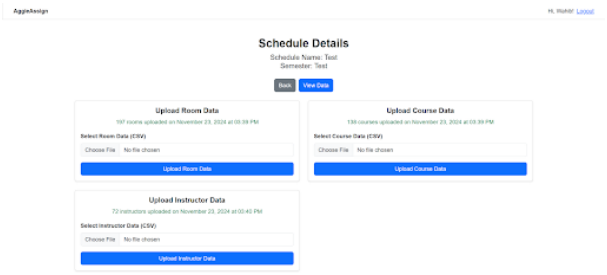

Fig. 1: User interface for data upload

instructor affinity for particular courses or timeslots. This data is usually retrieved from a Google Form sent to faculty every semester. If this data is not supplied, we assume instructors are ambivalent.

We require the above to be available in CSV format, which can be easily uploaded to AGGIEASSIGN. Our user interface for data collection can be seen in 1.

B. Resource View

Once the data is successfully uploaded, the user has the ability to view data in an organized manner. Each resource is given its own tab, which can be visualized in a tabular format, as seen in 2. This permits the user to take stock of

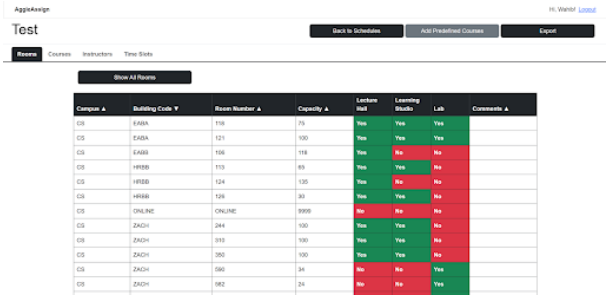

Fig. 2: User's view of all available courses

all available data and make corrections of any errors. When the user is satisfied, they can proceed to the *Add Predefined Courses* tab to begin generation.

C. Schedule Generation

Before creating a schedule, the user has the option to *lock* courses. This allows the user to fix a particular class, instructor, and room pairing to a given timeslot. This request must be respected by the scheduler, who will then attempt to generate

the best schedule containing the locked courses. Once the user is done locking courses, they can proceed with generation. The completed schedule will then populate the view and can be exported to a CSV file.

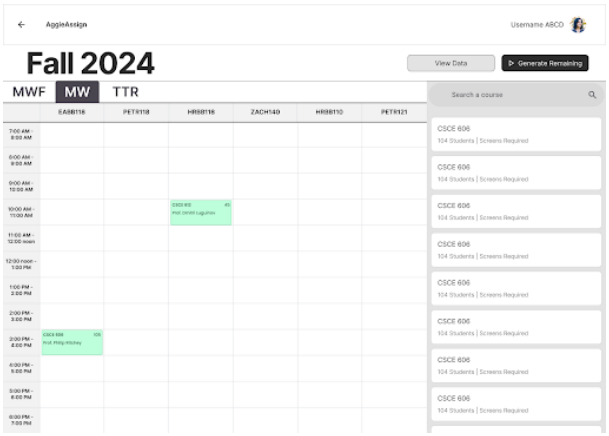

Fig. 3: A view of a schedule with two locked courses

III. Algorithm Implementation

In developing the algorithm, we classify constraints as either *hard* or *soft*. Hard constraints must be satisfied to create a valid schedule, while meeting soft constraints allows us to rank two different valid schedules. To attend to both of these requirements, we developed an algorithmic approach to schedule generation. This relieves the user of most of the burdens associated with schedule creation and should attain a more optimal solution. Optimality, and the constraints the algorithm considers, will be discussed in greater detail shortly.

A. Constraints

Our algorithm notes the following *hard* constraints:

- H1) All courses have precisely one instructor, time, and room.
- H2) A room must have a capacity at least as large as the assigned course enrollment
- H3) No two courses can share the same room at overlapping times
- H4) Locked courses are respected
- H5) Any course requiring special constraints (e.g. lab, learning studio, etc) gets a room meeting these criteria
- H6) Instructors cannot teach multiple classes in overlapping time slots
- H7) Instructors are not assigned more hours than their contract stipulates

Any schedule generated by the algorithm must abide by these requirements. If it is impossible to do so, the program will throw an error. We now mention *soft* constraints, which are determined by optional data about professor preferences.

- S1) Instructors are assigned courses for which they have a high affinity for
- S2) If an instructor dislikes early morning or late afternoon courses, they will be less likely to receive those assignments.

Unlike hard constraints, the scheduler is not obligated to fulfill these requests. The algorithm's performance in satisfying these constraints is provided to the user along with the generated schedule.

B. Algorithm Pipeline

We elect to partition the schedule generation into two parts. This decoupling was done to reduce the memory footprint of the application and to simplify the constraint logic. First, the courses are placed into rooms and times in a manner that minimizes the total number of empty seats. After this assignment is complete, we then pair professors to courses. This matching is done as to maximize the global professor happiness. For a professor p and class c, we define happiness as:

$$h(p, c) = A(p, c) + 5 \cdot \mathbb{1}\left[c \text{ is scheduled at a time } p \text{ likes}\right]$$

where $A(p, c)$ is the instructor affinity for the course. Affinity scores are integers in the range $[1, 5]$, with higher values indicating a greater desire to teach the course. This happiness function was chosen for its simplicity. Learning a better function via A/B testing is feasible, but outside the scope of this project.

C. Problem Formulation

Both of these assignments can be formulated as linear programs, which can be solved efficiently through the use of external libraries. Denote:

$$\begin{aligned}
\text{classes } C &= \{c_1, \ldots, c_i\} \\
\text{rooms } R &= \{r_1, \ldots, r_j\} \\
\text{time slots } T &= \{t_1, \ldots, t_k\} \\
\text{class enrollments } E &= \{e_1, \ldots, e_i\} \\
\text{room capacities } K &= \{k_1, \ldots, k_j\} \\
\text{locked courses } L &\subseteq C \times R \times T
\end{aligned}$$

1) Class Assignments: We can represent our schedule as a tensor X, where $X_{crt} = 1$ if class c is assigned to room r at time t and zero otherwise. This gives the following linear program:

$$\begin{aligned}
\min \quad & \sum_{c,r,t} x_{crt}(f_r - e_c) \\
\text{s.t} \quad & \sum_{r,t} x_{crt} = 1 \quad \forall c && \text{(H1)} \\
& x_{crt}(f_r - e_c) \geq 0 \quad \forall c, r, t && \text{(H2)} \\
& \sum_{c} x_{crt} \leq 1 \quad \forall r, t && \text{(H3)} \\
& x_{crt} = 1 \quad \forall (c, r, t) \in L && \text{(H4)}
\end{aligned}$$

where the above are formalization of the first hard four constraints.

2) Instructor Matching: Now that classes have been assigned times and rooms, we can accurately compute all possible instructor happiness scores. Like before, let X be a matrix such that $X_{pc} = 1$ if professor p is assigned to class c and zero else. Additionally, take h_i to be the maximum contracted hours for professor i and $O(c)$ to be the courses occurring at the same time as c. We have the following linear program:

$$\begin{aligned} \max \quad & \sum_{p,c} x_{pc} \cdot h(p,c) && \text{(S1 + S2)} \\ \text{s.t} \quad & \sum_{c' \in O(c)} x_{pc'} \leq 1 \quad \forall p && \text{(H6)} \\ & \sum_{c \in C} x_{pc} \leq h_p \quad \forall p && \text{(H7)} \\ & \sum_{c \in C} x_{pc} \geq 1 \quad \forall p \end{aligned}$$

where the final constraint is added to ensure that every professor is assigned at least one course.

D. Evaluation Metrics

For our final schedule, we use total professor happiness as our metric. This is compared against a "perfect" schedule, in which every instructor has a maximum happiness score for all of their assigned classes. More formally, if $C(p)$ gives the set of classes assigned to professor p, we have:

$$\text{score} = \frac{\sum_{p \in P} \sum_{c \in C(p)} h(p,c)}{\sum_{p \in P} h_p \cdot \max\limits_{c \in C} h(p,c)}$$

Note that this ideal schedule may be impossible, for a variety of reasons. Professors vying over the same classes or not ranking enough course highly, for instance, can make this score unattainable. Nevertheless, this provides a reasonable baseline for assessing our model's performance.

IV. Conclusion

A. Experimental Results

The full source code for AGGIEASSIGN can be found here. Our application was tested on anonymized data we received from the client. We were able to generate multiple valid schedules, with an 80% average satisfaction rate. Due to its dependence on a likely impossible perfect schedule, this metric induces an artificial ceiling on performance. We suspect that, outside of changing evaluation criteria or happiness functions, very little improvement can be made in performance. Execution speed was also reasonable, taking less than thirty seconds to generate the full schedule on a standard laptop.

B. Future Work

Though AGGIEASSIGN provides a great starting point, there are other intricacies associated with scheduling that we do not consider. A few examples include:

- Assigning online classes that do not have times or rooms
- Considering walking distance between buildings if a professor is assigned consecutive classes
- Handling lab courses that are shared across multiple sections
- Scheduling "exotic" courses, which require timeslots other than the standard 50 or 75 minute allocation

These points could be added in future iterations, without making any major changes to the scheduling algorithm. There are also potential gains to be made in application performance. Uploading professor affinity data is prohibitively slow, which can likely be improved by optimizing the SQL queries. Lastly, instructor input would be helpful in developing a more authentic happiness criteria. Our model gives equal weight to class time and material, but each particular instructor likely has a different balance in mind.

C. Conclusion

AGGIEASSIGN provides a user-friendly experience for generating a department's course schedule. Our upload scheme allows the user to supply data without the need for extensive formatting and offers visualization of all of the key resources. The core algorithm produces a schedule optimized for professors in a timely manner. Though AGGIEASSIGN doesn't capture all of the minutiae associated with scheduling, we believe it provides a valuable starting point, thereby eliminating days of manual work.