Adaptive Nested Algorithms for Balanced Scheduling

Stetson Bost
Department of Mathematics
Harvey Mudd College
Claremont, CA 91711
sbost@hmc.edu

Weiqing Gu
Department of Mathematics
Harvey Mudd College
Claremont, CA 91711
gu@hmc.edu

Abstract

Scheduling, adaptive algorithm, nested algorithm, machine learning, big data

1 Scheduling Problem

Scheduling is a very large component of modern life. In a fast-paced world, it can be important to effectively plan one's schedule in a way that is conducive to productivity while prioritizing the importance of a healthy balance between different types of activities.

We developed algorithms that create balanced schedules that can be adapted to person to prioritize his or her needs.

2 Background and Motivation

The primary motivation for pursuing this work was the perpetually active and hectic environment of a college campus, with Harvey Mudd College in mind. Students are faced with many academic obligations, from doing homework to attending class to studying for exams. While focusing on academics is important and necessary, it is also valuable to have a healthy balance of different activities to support a student's personal wellbeing. There are many ways for students to arrange their schedules, but rarely does a "one size fits all" approach work for creating students' schedules. Each person has a unique style of working and individual needs, so it makes sense that a person's schedule should be tailored to fit the person.

Additionally, schedules are inherently dynamic. Events can be scheduled and canceled, an individual may occasionally have to deal with an emergency that preclude all other scheduled activities, and priorities can change frequently.

At first glance, one might consider algorithms that solve Job Shop Scheduling (JSS) problems, which comes from operations research, but most algorithms for JSS are primarily concerned with optimization. This poses a problem for actual human schedules, because the "optimal" solution might not result in the most healthy or balanced schedule when people actually attempt to follow it.

There is a need for algorithms that are capable of creating balanced, adaptable schedules. We wanted to create algorithms that could do this in a way that tries to capture human intuition through algorithmic and machine learning techniques.

3 Examples of Scheduling Problems

There are many examples of personal scheduling problems that could be solved using similar algorithms. We list a few such examples here.

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

- One example, which will be explored later in this paper, is a college student making a weekly schedule that includes classes, homework, meals, free time, a job, sleep, etc.
- Another example might be an (office) employee's work schedule. This can be considered on different time scales. For a day, the employee might need a schedule that includes when he or she should start and end, what time different tasks or meetings will take place, and when to have meals and breaks. With a longer timescale, such as a month or quarter, the employee might need to schedule which days to work on different projects, when training or business trips will take place, and when to take personal time off from work.
- An elementary school teacher has required standards that his or her students must meet. The teacher must create a flexible schedule for the academic year that plans when to cover different topics in order to meet the requirements. Additionally, the teacher may need to plan for field trips, "slack" days (in case the class falls behind schedule), as well as incorporate a larger school-wide schedule including events such as assemblies, parent-teacher conferences, and breaks. The teacher may divide the topics into units, and the calendar into months or weeks. Depending on the pace of the class throughout the year, the schedule may need to be adjusted to either move faster or slower later on.
- A working parent may be at their workplace for a large portion of the day, but they also need plan their schedule to balance their work with family time each day.
- A family on vacation may have a list of activities they want to do while visiting a particular place before they leave.
- The organizer of a conference may have a list of speakers that are going to give presentations. The organizer wants to schedule the speakers and breaks (for snacks, meals, etc.) such that the audience members generally remain engaged throughout the conference.
- An academic department at a college may need to schedule regular tutoring session for several classes so that most students in those classes and all the tutors can attend. The department may also have special events on occasion, such as guest lecturers, that must be scheduled such that attendance is expected to be high enough to fill a lecture hall.
- A manager at a company may need to schedule weekly meetings with some of her employees as well as a meeting to make a special announcement with a large group of employees in attendance.
- A doctor's office may have many patients who want or need to schedule appointments. Some need annual checkups, some want to come in to discuss a recently developed pain, and others may urgently need medical help from their doctor. The office must give priority to the most urgent issues and make sure those patients get scheduled as soon as possible.

4 General Approach

For the remainder of this paper, we will use the terms *events* and *tasks* interchangeably to refer to an activity of some duration that should be added to a schedule. Their roles in the scheduling algorithms will be identical, but their use will depend on how they are used in typical English.

To address scheduling in a way that took into account balance and personal preferences, we wanted to use machine learning and mimic human intuition when possible. In general, humans will schedule regular (recurring) events first, then schedule events from high priority to low priority. Humans seem to intuitively prioritize tasks, but the ways that they do so can vary between different contexts and from person to person. Therefore, we wanted to create algorithms that were *adaptive*, in two senses of the word. First, we wanted something that could take in a stream of tasks to schedule and *adapt* according to when tasks are introduced. Second, we needed an algorithm that could *adapt* to an individual, taking into account the person's preferences and way of working.

We decided to use *nested* algorithms that assigned events to a schedule in stages, where earlier stages schedule events more generally (i.e. the week it will be scheduled), while the later stages arranged events more specifically (i.e. the exact time slot when the event should take place).

5 Nested Algorithm for Personal Scheduling

We created an algorithm that aims to use the power of machine learning to make a user's personal schedule. We wanted our algorithm to schedule the tasks that the user has specified while promoting a balanced lifestyle. The idea of a *balanced lifestyle* can vary greatly from person to person. Therefore, we need an algorithm that can adapt to the type of balance that suits the individual.

We will be using a college student's daily schedule, but this can be generalized to other situations that involve schedules. In order to have balance, it is valuable to have both work and non-work activities in the schedule.

Necessary information for this algorithm is the time during which tasks can occur (i.e. times when the person will be awake) and a stream of tasks to be completed. Each task has some additional information associated with it:

- deadline to complete the task or exact time frame to complete task (i.e. if the task is a scheduled event such as a lecture),
- estimated amount of time needed to complete the task,
- intensity level (a rough proxy for amount of concentration needed to effectively perform the task), and
- whether the task is work or non-work.

In order to get the intensity level, this can either be a user-inputted setting, or it can be learned from historical data (provided by the student) using a variety of machine learning algorithms.

To help effectively describe the algorithm, we will use the example of a typical Harvey Mudd College student during their second semester, which was the example used to originally develop this algorithm. Consequently, we will be using language consistent with that of a college student who is taking *classes*, doing *homework*, preparing for *exams*, etc.

A schedule is made up of time slots. For our algorithm, a *time slot* refers to the smallest period of time the algorithm will be dealing with. This can be 1 minute, 15 minutes, 1 hour, 1 day, etc. The schedule is preliminarily divided into *blocks* made up of some number of time slots. (For our example, we use four- and five-hour blocks.)

In order to have balance, each block is required to have some (minimum) amount of non-work time (i.e. time when a student is not doing academic-related activities). This time may be as structured or unstructured as the user desires, allowing flexibility to adapt to the user's preferences. Examples of activities that can occur during non-work time include eating, exercise, spending time with friends, napping, etc.

The algorithm consists of three nested steps.

1. Distribute the tasks into sections of the schedule (on a scale larger than blocks), where all tasks within a section will be completed by the end of section. The goal of this step is to make sure all tasks are scheduled to be completed by their deadlines.
2. Assign tasks in each section to a block. The goal of this step is to maintain a balance of intensity levels across the blocks within a section.
3. Schedule tasks within each block. The goal of this step is to adapt to the user's preferences.

We need to start with a list fo assignments sorted by due date, with the soonest deadline first. If multiple assignments have the same deadline, then sort those assignment by intensity, with the most intense first. For assignments with the same deadline and intensity, sort them by the number of hours remaining to complete the task. After this, the order is arbitrary.

5.1 Assign Tasks to Sections

A *section* refers to a time period that ends in an upcoming deadline or due date for which there are still tasks to be completed. There is a list of blocks that belong in each section.

The goal of this step is to take a list of tasks and assign some of them to the next available section.

We want to keep track of the current day (start of the section) and the next deadline (last day of the section). At the beginning of the algorithm, the start of the section should be specified by the user.

1. Determine the next deadline, which can be done by looking at the deadline of the next unscheduled task (which should be next in the task list). This is the *current section*.
2. Create a list of all blocks in the current section, and from these blocks, determine how much total available work time there is in the section. This amount should exclude scheduled events and non-work time.
3. Following the order of the task list, assign as many tasks as possible to the current section.

At the end of this, there should be a list of all the tasks that are going to be scheduled for the current section. At this point, we proceed the the next step of assigning tasks to blocks. This is repeated to determine the tasks in subsequent sections.

5.2 Assign Tasks to Blocks

Once a section is full, it is time to assign subtasks to specific blocks. One of the goals here is to respect each block's minimum allowable amount of non-work time. Additionally, we want to distribute work across the blocks in a section as evenly as possible to have a balanced schedule.

The term *subtask* here refers to a portion of a (possibly) longer task, as the duration of a subtask in a block may be shorter than the original task's duration in order to make it possible (and potentially easier) to assign subtasks to blocks. A note on subtasks: it is helpful to divide tasks into subtasks of minimum allowable duration (ideally to the time slot level) to ensure that it is possible to schedule all the subtasks.

To monitor balance, each block has an associated intensity level, which is a sum of the intensities across all time slots in the block. Note that even before this step starts, it may be possible that some blocks have nonzero intensity due to already-scheduled events.

This step of the algorithm goes through the list of subtasks, which is sorted by duration (longest first), then by intensity (highest intensity first), then by deadline (soonest first). In general, the block with the lowest intensity level is assigned the next subtask in the section's subtask list. Note that the minimum amount of non-work time must be respected in this step so that a block is not overloaded. If the subtask cannot be assigned to the block with the lowest intensity level (which can happen due to the duration of the subtask), then the subtask will instead be assigned to the next block of lowest intensity that *can* be assigned the subtask.

Once this is completed, all the subtasks should be assigned to blocks, and we can proceed to the final step of the algorithm.

5.3 Assign Tasks to Time Slots

This step can be done however the user prefers, using any scheduling algorithm that can assign subtasks to time slots in a helpful way. A requirement for this algorithm is that is must respect events that must occur during specified times, such as meals.

If the user desires, this step can be omitted, allowing for flexibility in how the subtasks are distributed within the block.

6 Algorithms for Event Scheduling

This algorithm schedules events that are expected to have multiple attendees. One of the goals is to schedule the desired event at a time when enough people are expected to attend.

6.1 Probabilistic Approach

Suppose we know the probabilities for all potential attendees that they will attend all possible events. This means we can know how likely they are to attend a new event if it is scheduled for a particular time slot. Let A be the list of attendees, t be a possible start time, $p_a(t)$ be the probability that an

attendee a will attend the event if scheduled at time t, and $\mathbb{E}[t]$ be the expected value of attendees at time t. Notice that for any particular start time t, if we sum all potential attendees' probability of attending, we will get the expected value of attendees at t. For each potential attendee a, go through all possible start times t, add the probability $p_a(t)$ of a attending at t to the value in the index corresponding to t. That is,

$$\mathbb{E}[t] = \sum_{a \in A} p_a(t).$$

If there are m possible start times, then we can create a $1 \times m$ matrix $\mathbb{E}$ of the expected number of attendees at each time. This matrix is calculated as

$$\left[\sum_{a \in A} p_a(t_1) \quad \sum_{a \in A} p_a(t_2) \quad \cdots \quad \sum_{a \in A} p_a(t_n) \right]$$

for all $1 \leq i \leq m$.

Determining the probabilities for attending the events can be learned from modifications to the classic machine learning k-Nearest Neighbors. However, this method may be too slow, so the following algorithm may be used instead.

6.2 Nested Approach

This follows the the personal scheduling algorithm described previously, with only slight adaptations.

Each event is considered a task where the constraints include the times when the event can (or cannot) occur, required attendees, priority in scheduling the event. There may be additional constraints on the schedule itself that correspond to maintaining balance. These can include a minimum or maximum number of special events over a time period and constraints on the frequency of events.

7 Data

We used data from the American Time Use Survey (ATUS) to test our algorithms. In order to extract the most useful items in this data set, we used the ATUS-X extraction tool, provided by IPUMS for their Time Use project.

Acknowledgments

We would like to thank the Rose Hills Foundation for their generous support of this research.

References