Scheduling Professional Development with SAT Solver

Rosalinda Garcia

CS517

1 Introduction

In the realm of education, professional development is utilized to help faculty members learn and improve the skills they implement in their classrooms. However, one of the trickiest aspects of planning and executing a professional development program is the scheduling. This is due to the fact that faculty have complicated schedules with different courses at different times of day during different school terms.

The motivation for considering this problem was a recent research study I conducted that required the scheduling of a professional development workshop for CS faculty. Additionally, the workshop is now being updated and scheduled again, bringing this problem to the forefront.

In this paper, I will describe the problem of professional development and how it Karp reduces to SAT. Then, I will describe how my new tool implements this reduction and utilizes pySMT to interface with MathSAT to solve instances of this problem.

2 Related Work and Motivation

The primary motivation for this work was my 2024 study about a professional development workshop for faculty [1]. In this work, my co-authors and I present a new approach to teaching inclusive design in CS classrooms. To implement the approach requires participation and collaboration from many faculty members in a professional development workshop.

While this first iteration of the workshop was successful, scheduling it so that all of the faculty attendees could benefit from it was difficult. In fact, the workshop was actually held twice due to scheduling conflicts.

This year, the scheduling issue has resurfaced as we try to find a time that works for our updated workshop. This is especially difficult as we are unable to schedule two workshops this year. Thus, we must maximize the outcomes from a singular instance.

Current solutions for this problem include scheduling polls like Doodle or Rallly. Participants provide their availability on these polls and then it is up to the organizer to determine which time is best. While this is functional, it can be hard for the organizer to pick the best time (especially with larger groups of participants) and the burden still rests on the human user.

With my tool, I hope to place the burden of solving the problem on the SAT solver.

3 Methodology

3.1 Defining the Problem

As I mentioned before, the problem this tool aimed to solve is the issue of scheduling professional development events. As such, I determined that the tool would need the following input:

- t: The length of the event as an integer number of hours
- filename: The name of a file holding a 2D array that give participants' availability
- k: the minimum number of attendees required to be at the event at any gives time

The 2D array in the file indicated by the filename variable is of the following format: each row indicates a faculty member and each column indicates an hour in the 8 hour day from 9am-5pm. Thus, the values tell a particular faculty member's availability at a given hour. For example an array for four faculty members might look like:

$$\begin{matrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \end{matrix}$$

This input format is very similar to what might be input to a scheduling poll application except it is represented in 1's and 0's instead of buttons or a list of hours. There is no "standard" input format for a scheduling problem like this as availability can be represented in many different ways. However, I believe that this 2D array is a simple and understandable representation that we can easily translate to and from other representations.

The time length t of the event and the faculty's availability are required for solving any kind of scheduling problem. I added k because I have found in my work that it is rare to have all faculty attend the entire duration so this provides the option for faculty to miss part of the event while maintaining a mostly full room.

These variables came with a few constraints:

(1) **The chosen time period of length t should be one solid block of time.** (For example a 4 hour period might be 9am-1pm but not 9am-10am and 11am-2pm.) This constraint was added because the majority of professional development requires consistent discussion or work that is hard to do over broken up periods of time.

(2) **All faculty must attend at least t-1 hours of the event** (if the event is more than 1 hour long). This constraint was added because while we expect faculty to not be able to attend the full event, it is important that they attend the majority in order to successfully learn the content of the professional development.

(3) **At least k faculty must be available to attend at any given time.** As described above, it is rare to have faculty be able to attend the full event, so k provides a minimum number faculty who are in attendance at any given time so that the room is never empty

I refer to these three constraints as Rule 1, Rule 2, and Rule 3.

3.2 Generating SAT Formulas

In order to solve instances of this problem, I created a tool to turn these professional development scheduling problems into SAT problems. I began by defining a complete day as an 8 hour period of time from 9am to 5pm. I then defined a literal for each of the 8 hours as h1-h8. The assignment of these literals to satisfy the ending formula will provide a solution to the scheduling problem. Additionally, this keeps the number of literals from expanding, causing it to be more inefficient. To create formulas, I had to determine a method for generating formulas to ensure each rule described previously was met. The method for each rule follows below.

3.2.1 Rule 1: The selected time is one time period of length t hours.

Suppose that we want a time window of length t hours that begins at some time X. Also, consider that a one hour window that begins at time X is represented as hX. To ensure that the window of length t is one continuous period, we can ensure the following is true:

$$\alpha_x = (hX) \wedge (h(X+1)) \wedge \ldots \wedge (h(X+t-1))$$

For example, if we want a 4 hour window that starts at hour 1 (h1) we might have:

$$\alpha_1 = (h1) \wedge (h2) \wedge (h3) \wedge (h4)$$

This assures that each hour following hour X is true until there is a length of time t.

Now, we also know that there are 8-t+1 options for time periods of length t in the 8 hour day. So, there's a possible time window that begins at each hour from hour 1 to hour (8-t+1). Additionally, each possible window has an α_x that needs to be true if the window is chosen.

However, only one of these time periods will be chosen (even if there are multiple solutions) so only one α_x will be true. Thus, we take the "or" of each of these α_x which results in the following:

$$\gamma_1 = \alpha_1 \vee \alpha_2 \vee \ldots \vee \alpha_x$$

This γ_1 is the formula for rule 1.

3.2.2 Rule 2: Each hour has at least k faculty.

Consider that if we were to check every option for hX, we would have to check (8-t+1) windows and see if k faculty are present for every window. This would expand very quickly. However, the input 2D array is designed so we do not need to expand all the way. Instead, there is a trick to simplify the formula creation.

Recall that in the 2D array, each column is an hour with the values in the array indicating 1 if the faculty member is available at that hour and a 0 if not. We can take the sum of the values in each column of the array and check if the number of faculty available is greater than or equal to k.

Then, if there is an hour that has fewer than k faculty, we know the solution does not include that hour. So, we add the complement of the literals for these hours to a formula as follows (where hX, hY, and hZ are hours with less than k faculty available):

$$\gamma_2 = \neg hX \wedge \neg hY \wedge \ldots \wedge \neg hZ$$

This γ_2 is all we need to ensure Rule 2 is met.

3.2.3 Rule 3: All faculty must attend at least t-1 hours

At first glance, the way to check this rule also seems like it should expand all possible options to check which ones meet this criteria. However, there is no need to do this as it would expand very very quickly. Instead we can leverage the algorithm from rule 1 and look at unavailable times instead of available times.

For this rule, the tool first checks to see if each faculty member has any unavailable times. If they are available for all times, then there is no need to enforce this rule for them.

If a faculty member has unavailable times, then it must check to see if the unavailability results in any time windows of length t where the faculty member is unable to attend t-1 hours.

To do so, the tool looks at each time window of length t, as it did for rule 1. However instead of checking that each hour is true, it checks to see how many of those t hours the faculty member is available for. If it is less than t - 1 hours, then this time window is added to a list of unavailable options. This looks similar to how the rule 1 formula was built but each term has a “not” operator because these are unavailable times and it uses the OR operator because we want the complement of this time window:

$$\beta_x = \neg(hX) \vee \neg(h(X+1)) \vee \ldots \vee \neg(h(X+t-1))$$

For example, if we are looking at a 4 hour time window beginning at hX and a faculty member is unavailable for the first two hours, we would add the following to a list:

$$\beta_x = \neg(hX) \vee \neg(h(X+1)) \vee \neg(h(X+2)) \vee \neg(h(X+3))$$

Once the tool has checked the availability for this one faculty member, it should have a list of all the time windows that are unavailable to this faculty member. If there are multiple time windows in this list, the tool takes an AND of these options so that we know the constraint is met regardless of the selected window. So we have the following for each faculty:

$$\delta_x = \beta_1 \wedge \ldots \wedge \beta_n$$

As the tool continues, it makes one of these formulas for each faculty member. However, we want to make sure that all faculty avoid their unavailable time windows. Thus, the tool takes an AND of each of these δ formulas. As a result we have the final formula for this rule as:

$$\gamma_3 = \delta_1 \wedge \delta_2 \wedge \ldots \wedge \delta_n$$

3.2.4 Putting it all together

In order to represent the full problem, we want each of the rules to be met. Thus, we take the "and" of each of the formulas for the three rules. The result is:

$$\gamma_1 \wedge \gamma_2 \wedge \gamma_3$$

Thus, by finding a satisfying assignment to this formula, we have solved the scheduling problem.

3.3 Interfacing with pySMT

To prepare for writing this tool, I installed pySMT and used the built-in pySMT installer to install MathSAT.

Then, I used Python and the pySMT shortcuts (e.g. "And", "Or", etc) to create the formulas described above that ultimately came together as $\gamma_1 \wedge \gamma_2 \wedge \gamma_3$.

Once this formula was ready, I input it to MathSAT using pySMT's get_model function. This function returned a model object. If it is non-empty, the tool prints out the model which says which hours are "true" (part of the scheduled time) and which hours are "false" (not part of the scheduled time). An example of this output is given in Section 4. If the model is empty, then the tool prints out "No solution."

3.4 Test Cases

In order to test the tool, I needed a few test cases. I originally wanted to use the real set of faculty availabilities from our first workshop. Unfortunately, these data do not exist anymore. Instead, I took availability from another real source: the scheduling polls from my lab meetings. Our lab meetings are scheduled each term with approximately 15 people each term. This is a similar size to the workshop scheduling problem as there were 18 total faculty in my study [1].

I also created a test case with a larger participant count by concatenating the scheduling polls.

4 Results

4.1 Example Output

Test case 1 was a set of availability from my lab meeting scheduling poll with 14 participants:

```
1 1 1 0 0 1 1 0
```

```
0 1 1 0 0 1 1 0
0 1 1 0 0 1 1 0
1 1 0 0 0 1 1 1
1 1 1 0 0 0 0 1
1 1 0 0 0 0 1 1
1 1 0 0 0 0 1 1
1 1 1 0 0 1 1 0
0 1 1 0 0 0 0 0
0 0 1 0 0 1 1 1
1 1 1 0 0 1 1 1
0 1 1 0 0 1 1 1
0 1 1 0 0 1 1 1
1 1 1 0 0 1 1 1
```

When run on this input and asked to schedule a 2 hour meeting with a minimum of 10 attendees at a given time, the tool output the following:

```
h1 := False
h2 := True
h3 := True
h4 := False
h5 := False
h6 := False
h7 := False
h8 := False
```

This means that hours 2 and 3 were selected for the meeting time. In other words the meeting should be from 10am-12pm.

If this tool were to be expanded, it would be helpful to convert this output format to a more human-readable format. This might even just be done by replacing h1-8 with 9am-4pm or creating a visualization of the schedule.

4.2 Time and Costs

For the test case shown above, the calculation took ~0.017 seconds. This is for approximately the problem size that this tool was built for. In the context of a user wanting a quick answer about scheduling, it certainly finished in a reasonable amount of time. Unfortunately, I was unable to find a way to get pySMT to output the size of a given formula, so I stuck to time costs.

Despite the smaller participants numbers being more close to real life scenarios, I still tested with larger data sets and with various input parameters.

In terms of the number of total faculty participants, the time needed scales proportionally. For 28 participants with a minimum of 20 attendees (double the participant count and double the minimum participant count) it took ~0.019 seconds. For 56 participants with a minimum of 40 attendees present, it took ~0.021 seconds. For 224 participants with a minimum of 160 participants present, it took 0.026 seconds. Logically, this makes sense: increasing the number of participants increases the number of checks that need to be done in order to verify rules 2 and 3. So we see a proportional increase between number of faculty participants and time cost, but it is not enough that a user would find it problematic.

Additionally, the length of the event is inversely proportional to the time cost. For example, asking for a 1 hour time window with minimum 10 participants out of a total of 14 participants took ~0.019 seconds while asking for an 8 hour window on the same input took ~0.017 seconds. When looking at the algorithm, we see that this parameter greatly affects rules 1 and 3 and that is the reason for this change in cost. In other words, as the length of the event shortens, there are more possible time windows that need to be checked. Still, this difference in cost is not likely to be problematic for a user.

However, it does not seem that the minimum number of faculty has much effect on the time cost. This makes sense as this parameter is only involved in rule 2 and the computations for rule 2 do not rely heavily on the size of the minimum number of faculty. Instead, they rely more on the total number of faculty.

Perhaps if someone tried to use this tool on an event for thousands of faculty, it would take too long, but for the problems that it was created to solve, it has an efficient running time.

Conclusion

Overall, using a SAT solver to help support scheduling professional development is a useful and efficient technique. This allows the burden to be placed on the SAT solver instead of leaving the solution up to a human who may struggle if there are large numbers of attendees.

Additionally, this type of scheduling tool may be helpful in other contexts as well. For example, I used lab meeting scheduling as a test case and this might be another useful application of the tool.

Works Cited

[1] Rosalinda Garcia, Patricia Morreale, Gail Verdi, Heather Garcia, Geraldine Jimena Noa, Spencer P. Madsen, Maria Jesus Alzugaray-Orellana, Elizabeth Li, and Margaret Burnett. 2024. The Matchmaker Inclusive Design Curriculum: A Faculty-Enabling Curriculum to Teach Inclusive Design Throughout Undergraduate CS. In Proceedings of the CHI Conference on Human Factors in Computing Systems (CHI '24). Association for Computing Machinery, New York, NY, USA, Article 657, 1–22. https://doi.org/10.1145/3613904.3642475

Appendix

GitHub Repository: https://github.com/roseg31/CS517

PDF of code attached beginning on next page.