Algorithm Implementation and Analysis for the Great Explorations Matcher and Scheduler

Christopher James Pyles

An advanced project report submitted in partial fulfillment of the requirements for graduation with Honors in Computer Science.

Whitman College
2020

Certificate of Approval

This is to certify that the accompanying advanced project report by **Christopher James Pyles** has been accepted in partial fulfillment of the requirements for graduation with Honors in Computer Science.

John Stratton

Whitman College
May 21, 2020

Abstract

Our Capstone group's clients are tasked every other year with curating matches by hand that lead to high student satisfaction with their workshop event. As this workshop event has expanded from 50 attendees in 1984 to over 400 in 2019, the need for a more efficient approach has become necessary. I took the lead role in my Capstone group in designing an algorithm that could match attendees with workshops to maximize satisfaction and efficiency. This paper outlines the problem in detail, the difficulty of such a problem, the approach taken to solving it, as well as some areas where the algorithm could be improved.

The source code for this project can be found at `https://github.com/WhitmanCSCapstone/ge-scheduling`

1 Introduction

Every other year, Whitman College hosts the Great Explorations workshop event; an opportunity for middle school girls interested in STEM subjects to attend workshops and become inspired. Each time this event takes place, over 400 girls must be organized into 30 different workshops at 3 different timeslots. Additionally, each student attendee lists their top 6 most preferred workshops when registering for the event. In previous years, it was the responsibility of Carol Morgan and Ruth Ladderud to choose which girls were assigned to which workshops by hand. The goal of my Capstone project's group was to automate this process, creating an efficient algorithm to match girls with workshops and time slots, working within the constraints laid out by Ruth and Carol.

2 The Problem

Due to the nature of the problem of matching students, workshops, and time slots, we hypothesize that this problem is NP-complete. While I have been unable to formulate a proof for its NP-completeness, it shares enough similarities with other NP-complete problems such as 3D matching and set cover that I can confidently make the conjecture that this problem is NP-Complete.

It is worth noting that the version of this problem we are working on is further constrained. Specifically, workshops must also be filled to a minimum capacity, and each workshop has an equal number of open slots for each time. However, I do not believe this makes the problem no longer one that is NP-complete. This is because several other NP-complete problems, namely vertex cover and independent set, are still NP-complete when put under similar degree constraints.

For these reasons, I chose to design the algorithm as if I were solving an NP-complete problem. It is designed to be flexible on input, and can be used to solve the most general form of the problem at hand, which we know to be NP-complete.

3 The Algorithm

Listing 1: The pseudocode of the Algorithm

```
For each student:
    For each preferred workshop:
        Increase the workshop’s popularity score by 1

Sort the workshops from least to most popular

For each workshop:
    For each preference rank, starting with the highest:
        For each student:
            If the student’s preference at the current rank is the same
            as the workshop, and the student is eligible to be assigned
            that workshop:
                Assign the student to the least full workshop session
                possible
                Check to see if the workshop reached minimum capacity. If
                it has, break and move on to the next workshop

    If all preferences have been examined and the workshop has not yet
    reached minimum capacity, do the following:
        While the workshop has not reached minimum capacity:
            Choose a random student not chosen before for this workshop,
            and not given a filler workshop previously. If the student is
            eligible to be assigned to the workshop:
                Assign the student to the least full workshop session
                possible as a filler workshop
                Assign the student to their most preferred workshop
                possible
                Check to see if the workshop reached minimum capacity. If
                it has, break and move on to the next workshop

Once all workshops have reached minimum capacity, do the following:

For each student:
    If the student has any empty time slots:
        For each preference rank, starting with the highest:
            If the student can be assigned to the workshop of the
            preference rank, assign the student to it.
        While the student still has empty time slots:
            Assign the student to a random eligible workshop
```

The algorithm that handles the matching and scheduling is a greedy algorithm. In other words, it is one that matches students to workshops and time slots as it runs, and commits to those matches rather than changing or reverting them later. The purpose of using a greedy algorithm is to come up with a final set of matches that we can consider "good" in an efficient amount of time. While it may not produce the best possible matches, doing so would be unrealistic in a reasonable amount of time.

The algorithm works by splitting the data into three major classes: one for students, one for workshops, and one for individual workshop sessions. There is also one matcher object responsible for the decision making of placing the students into workshop sessions.

The algorithm begins by determining a "popularity" score for each workshop. This is done by simply finding the total number of students who listed that workshop anywhere within their top 6 preferences. The workshops are then ordered from least to most popular, so that the least popular workshops will be filled first. This is to ensure that every workshop can reach a minimum threshold determined by a user-given input.

Once the workshops are sorted, the matcher goes about filling the workshops, starting with the students who prefer them the most. Beginning at the least popular workshop, the matcher searches the first preference of each student. If it matches the workshop being checked, and that student shares an available time slot with the workshop, then the student is assigned to that workshop. Additionally, the student is assigned to the least full workshop session possible, so that the sessions are filled with relatively even distribution. Once every students' first preference is checked, the matcher repeats the process with the second, third, and all subsequent preference rankings, either until the workshop reaches its minimum capacity, or until all student preferences are exhausted.

If all student preferences are exhausted, and the workshop still has yet to reach its minimum capacity, then the algorithm enters a second phase of matching students to workshops. Out of all students with two or more available open time slots remaining, one student is chosen at random. If possible, they are assigned the workshop, and then compensated by also being assigned the highest possible workshop on their preference list. If a student is assigned a workshop not on their preference list in this way, then it is recorded within that student object's fields such that the student cannot be randomly chosen to be given an unpreferred workshop again. This is repeated until the workshop reaches its minimum capacity.

The processes illustrated above are repeated for each workshop, moving up the lists of workshops from least to most popular. Once this process is completed, each workshop will be filled to its minimum capacity. However, not every student is necessarily assigned a workshop for every timeslot. The algorithm finally resolves this by choosing said students in a random order, and assigning them their most preferred eligible workshop. If no such workshop exists on their preference list, they are assigned one at random.

4 Runtime Complexity

The input to this algorithm is a set of student preferences of size N, and a list of workshops of length M. The runtime complexity of the various part of the algorithm are as follows:

The process of finding the popularity scores of each workshop and then sorting them by popularity is $O(N + MLog(M))$.

The process of filling each workshop with students who list them as preferences takes $O(N * M)$ time.

The process of filling each workshop to its minimum capacity with students who do NOT list them as preferences takes $O(N * M)$ time.

Finally, giving each unassigned student their highest remaining preferred workshop takes $O(N)$ time.

Altogether, the runtime of the algorithm can be simplified to the sum of these four runtime complexities. The most complex of these is filling the workshops with both students who prefer them and students who don't prefer them. Using this, we can simplify the overall complexity of the algorithm to be $O(N * M)$. Since the algorithm works on the most general form of the problem, and that form of the problem is NP-complete, a polynomial runtime of $O(N * M)$ is considerably good. The real-time runtime of this algorithm is just under 30 seconds, which is much better than the tens of hours that this process took when it was done by hand.

5 Quality of Results

After running the algorithm, the metadata of the results is outputted to a separate sheet. By reviewing these results for our test set, it's worth noting that unpreferred assignments were kept to a minimum, almost entirely contained within the workshops with popularity scores too low to be filled by students who preferred them.

Furthermore, every student in the test set was assigned at least two of their preferred workshops out of a total three. In fact, the vast majority of students had all three of their assigned workshops ones that they listed as preferences.

It is worth noting, however, that the algorithm's design does not guarantee this to be the case. It is possible for a student to be assigned only one of their top six preferences. This could happen if they are randomly selected to be given an unpreferred workshop to fill it to minimum capacity. Later, that same student could be given an unpreferred workshop during the final matching phase. However, this is only possible if all of the students eligible preferences are already filled, and that student was never placed in those workshops during early matching. For this to be the case, all six of the student's preferences would necessarily be high-popularity workshops, and a considerable number of other students would need to have similar or identical preference lists.

Unless all the above conditions are met, the student would necessarily be assigned a more preferred workshop during the earliest matching phase. Furthermore, given an input with such a student, there necessarily exist a nontrivial number of students with similar preferences. Therefore it would most likely require by-hand matching or an exponential-time algorithm to guarantee that no student would be given two unpreferred workshops, or to prove that such a matching could not exist.

I am also inclined to believe that this scenario, while possible, is unrealistic given the distribution of preferences we saw in our test dataset. Even workshops with popularity scores many times higher than their capacities did not cause any of the aforementioned problems. Furthermore, even students with preference lists consisting of mostly high-popularity workshops still received a minimum of two preferred workshops. Since the algorithm fills the most popular workshops last, the pool of students eligible to be assigned those workshops is greatly reduced by the time they are addressed.

6 Future Improvements

Despite the above analysis showing that the algorithm's outputs are reliably satisfactory, there is still room for improvement. For example, the algorithm does not guarantee that the final phase of matching will be able to assign students to their preferences. This could be improved by making minimal adjustments to previous scheduling or matches to allow a greater number of preferred workshops to be assigned. This could be accomplished either by rearranging a student's schedule, changing several students' assignments entirely, or a combination of both.

Additionally, there is a lot of room for improvement in how the algorithm handles scheduling. While the current system prioritizes putting students in the least full workshop sessions, it does not force or guarantee it. It is noticeable in the metadata output that there is not a very even distribution of students in the different sections of some workshops. This could be resolved by putting tighter restrictions on session filling. For example, each session could have its own minimum threshold that must be met, not just each workshop as a whole. Another possible solution would be to prioritize assigning students to workshops only when they can be assigned to the least full session, instead of simply when they can be assigned to any session. Treating the scheduling with higher priority would be an appropriate extension of this project.

7 Conclusion

Overall, the algorithm performs well, and effectively accomplishes the original goal of matching and scheduling students with workshops and time slots. For a problem that we hypothesize is NP-complete, the polynomial runtime is good, and the real-time runtime is exceptional compared to scheduling by hand.

As mentioned above, of course, there are several improvements to the algorithm that could be made. While we were unable to address these during this academic year due to time constraints, I hope that this algorithm can be improved upon in the future. While some of these changes would slightly increase the algorithm's runtime complexity, the algorithm runs quickly enough that it can be afforded to slow down the matching process to produce better results.

8 Acknowledgements

I thank John Stratton for the assistance in forming the conjecture of NP-completeness, as well as for his supervision of the entire project's development.

I thank Carol Morgan and Ruth Ladderud for the opportunity to tackle this difficult but engaging problem. I enjoyed having the chance to create a challenging algorithm that would benefit people in future years of this workshop event.

Finally, I thank my team - Ian Hawkins, Trung Vu, and Isaiah Standard - for assistance in producing the algorithm, as well as help producing the code that formatted the input and output of the system and its metadata.