Report: Assignment 2, Schedules

L. Schouten s2660148
D. Macquine s2592991

November 6, 2021

1 Introduction

This report is about an optimal algorithm and a greedy algorithm which are used to make a schedule for an educational semester. The optimal algorithm uses backtracking to find an optimal schedule. The greedy algorithm uses heuristics to find an sub optimal schedule, that is, the schedule found with the greedy algorithm may break certain rules but should keep the number of broken rules to a minimum.

The problem

For this assignment, it was necessary to create an educational schedule. For this schedule to be valid, certain constraints have to be met:
- All the courses should be offered exactly once.
- If a track follows a course on a certain time slot, that track cannot follow another course at that same time slot.
- When a track follows one course on a day, that track should follow at least two courses that day.
- On each day, a track may have at most one interval hour.

State and action

The state of the schedule is stored as an 3D string numpy array. The first dimension of the numpy array is for the different days. The second dimension is used to store the different time slots within one day and the third is used to store the different rooms that can be used on a certain time slot. The rooms within the schedule can either be empty or filled with a course. Courses have certain tracks that follow those courses. Courses are stored in a dictionary where the course is the key and the tracks that follow the course are stored in lists as value in the dictionary to that course. Actions can be described as filling an empty room with a course if allowed by the constraints, and removing a course from a room.

State space diagram: 1 weekday, 3 tracks and 3 courses

Figure 1 shows an example of a state space diagram for one weekday, three tracks and three courses. The three courses are the courses A, B and C. Course A is follows by the tracks 2 and 1, course B is followed by the track 0 and course C is follows by the tracks 0 and 2. The first room that is filled will always be room 1 of the first time slot by all of the possible courses. Initially the other rooms of other time slots aren't filled because this would cause duplicates. Furtheremore, it is assumed that rooms of a certain time slot are replaceable with other rooms of that same time slot and that therefore the room in which a course is placed does not matter.

Because of this, when courses could be given in different rooms, just one is chosen to continue from. Furthermore, only valid schedules are shown and schedules which can be created by illegal actions are therefore not shown. As can be seen from Figure 1, no schedule can be made for this example. The reason for this is that track 1 is only in one of the courses and whenever a track follows one course on a day, it should follow at least 2, which is impossible for track 1 here.

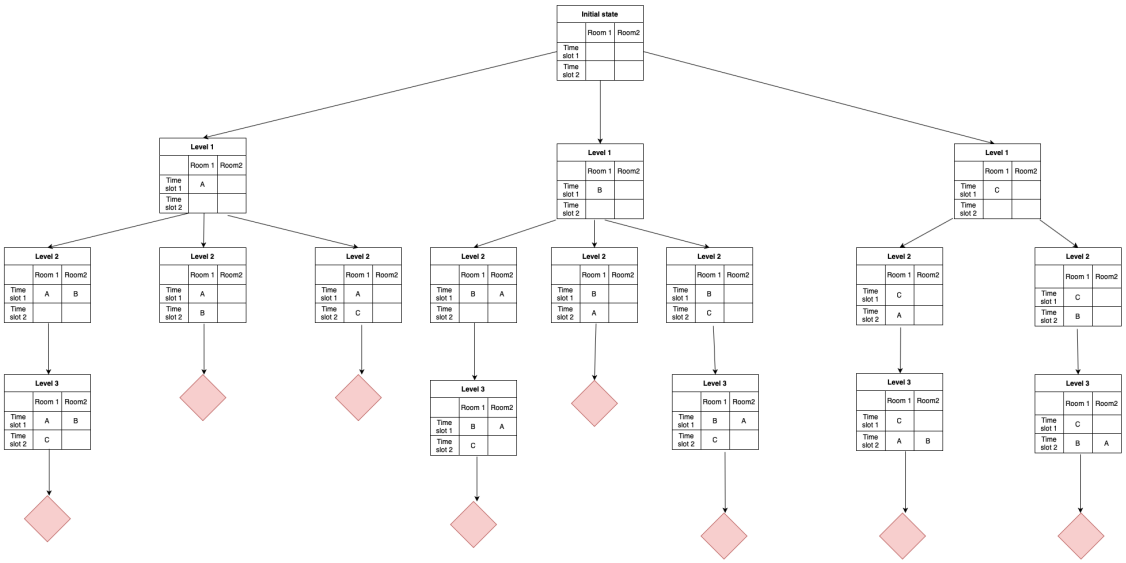

Figure 1: State-space of a one weekday schedule with two time slots and two rooms.

2 Analysis of the optimal method

The optimal method consists of a backtracking algorithm that iterates through all possible schedules and checks the schedule with each iteration for violations of rules (with the 'violaterules()' method).

When no rules are violated the algorithm decides to try all unscheduled courses in the next room in the next recursion. Once all courses are scheduled and the algorithm still complies with the rules that are checked in the 'violaterules()' method it runs the 'final_check()' method which checks the last rule: 'If a track has a course on a day it has to have atleast two courses on that day'. If the made schedule complies with the final rule the algorithm returns the optimal schedule, if not the algorithm returns to the previous recursion and tries other possible schedules.

Once a violation is found the algorithm removes the placed course and iteratively tries the other courses for the room. When the algorithm finds violations for every course on a certain room, it decides to leave that specific room empty and checks one last time for violations. If the rules are still violated the algorithm returns 'None' in which case the algorithm either returns to the previous recursion and tries another possible course for the previous room or it returns 'None' as the last return of the recursion which means no possible schedules exist that comply to the set rules.

To test the backtracking algorithm a unit test test was designed. This unittest is in the private_test.py file and is called 'test_build_schedule_backtracking'. This unittest uses 21 courses, 3 tracks, 3days, 4 time slots per day and 2 rooms on each time slot.

2.1 Optimal method on varying number of courses

How much time does the optimal method take on games with varying number of courses

-The runtime of the optimal method to create a schedule with 6 courses and 4 tracks with one weekday, 3 slots and 2 rooms is 0.002 seconds.
-The runtime of the optimal method to create a schedule with 22 courses and 3 tracks with 4 weekdays, 4 slots and 2 rooms is 0.106 seconds.
-The runtime of the optimal method to create a schedule with 52 courses and 7 tracks with 30 weekday, 30 slots and 30 rooms is 3 minutes and 20 seconds.

2.2 Limit that can run within 10 minutes

What is the limit that you can run within 10 minutes?

With more extreme parameters, eventually an recursion error occurs, even before the 10 minutes is reached.

3 Greedy Method

The greedy method "build_schedule_greedy()" tries to make a schedule based on the 'Most constrained variable' heuristic, to find the room with the least amount of course options to be placed for that room. After this room is determined, the best course that can be placed, without violating certain rules specified in the 'violatesrules()' method, will be determined. This is done by checking for each course that can be placed on the most constrained room on how many other rooms this course can be placed. The course that has the least amount of rooms it can be placed on will then be placed on the most constrained room.

Opposed to the backtracking method, the greedy method does not run the 'final_check()' method. This method checks if a track has atleast two courses on a day when it has a course on a day. Because of this, the greedy method doesn't always return an optimal schedule. Furthermore, when no course can be placed on the most constrained room without violating any rules, the Greedy Method will just return the schedule it build thus far. The reason to decide to stop building the schedule and just return the schedule as it is at that particular moment is that now the only two rules which might be broken are that not every course is in the schedule and that if a track follows one course on a day, it should at least follow two courses that day. It could be possible to continue building the schedule with for example "Monte Carlo tree search" however, by doing so, other rules will be violated. It can be argued that breaking some rules is less severe than braking others. If it is decided that breaking the other rules is less severe than the two just mentioned, Monte Carlo could be used.

3.1 How well does the greedy method work?

To test how well the greedy method works compared to the backtracking method we decided to apply the two unittest that were given for the backtracking method to the greedy method. For both methods; 'test_build_schedule_backtracking()' and 'test_build_schedule_backtracking_2courses_day()' the greedy method produces a sub-optimal schedule. We also made a unit test, 'test_build_schedule_greedy', which tests the greedy method to 21 courses.

In the first test ('test_build_schedule_backtracking()'), the greedy method finds a violation of the rules specified in the 'violatesrules()' method for one the courses. Because the greedy method cannot undo and redo a decision it finishes the schedule and returns the schedule without the course that couldn't be placed because of the violation. The downside of this is that one course is missing from the schedule, the upside is the 'Two courses that are offered to the same track are not offered in the same time slot' rule is not violated.

In the second test ('test_build_schedule_backtracking_2courses_day()'), the greedy method doesn't take into account the rule 'If a track has a course on a day it has to have atleast two courses on that day' and produces for that reasons a sub-optimal schedule.

The unittest 'test_build_schedule_greedy' tests if the greedy method holds the rules specified in the 'violatestrules()' method. The test is succesfull if this is the case meaning the greedy method produced a sub-optimal schedule. The greedy method succesfully runs this test.

3.2 Adverserial Testcase

The unittest 'test_build_schedule_adverserial_backtracking_greedy()' shows a case where the greedy method doesn't provide an optimal schedule and the backtracking method does.
In this test, course 'A' in has to be followed by track '2' and is placed on weekday 1 in the greedy method. Courses 'K' and 'L' who also have to be followed by track 2 are placed on weekday 2. So greedy method for this test case doesn't hold to the rule that is specified and would be checked in the 'final_check()' method, and the schedule returned is therefore not optimal.

The backtracking method places all the courses that have to be followed by track 2 ('A', 'K' and 'L') on the first day, while also complying with the other rules. Therefore the backtracking method does return an optimal schedule.

The reason as to why this final check isn't implemented into the greedy method is because the final check can only be done when a course is placed and the greedy method can't undo a made decision.

4 Summary

For this assignment we had to create a backtracking algorithm as well as a greedy algorithm to build an educational schedule taking into account certain rules. The backtracking algorithm determines if an optimal schedule can be found. Such a schedule has to take into account every rule. The greedy algorithm uses the 'most constrained variable' heuristic to determine a sub-optimal schedule. The downside of the backtracking approach is that it has to try every candidate solution for a schedule and backtrack to the last solution that was still valid if the current one isn't. The backtracking algorithm keeps doing this until a schedule is found which complies to all the rules. The greedy method can just iteratively build up the schedule and does

not have to check all the candidate solutions. Therefore the greedy method often finishes earlier. Therefore, especially on bigger schedules the greedy method might be the better one to use.

We also like to make some remarks about the process of this assignment. Unfortunately we weren't able to make a unittest that runs approximately 10 minutes for the backtracking algorithm. We don't know what kind of reason is behind this. In making our backtracking algorithm, we used the help of one of the TA's. We checked with him if our code seemed right and he thought this was the case. The supplied unittests ran without problems as well and therefore we did not have suspicions regarding our algorithm until we had to come up with a unittest that should take 10 minutes. We were not able to make such a unittest because the code seems to be running way too fast.