AI Project Report

CSP Formulation to Solve Scheduling Problem

Ahmed Wael Fouad - 201500862

Shrouk Mohamed - 201401023

MOTIVATION

One of the problems which require a lot of efforts in order to solve is coming up with a suitable schedule for a university, given a certain number of available rooms, a certain number of courses, and professor's schedules, it would require a lot of efforts, trials and errors, by the admissions office of university in order to form this schedule and satisfy all of the given constraints.

In an effort to make this problem easier, we introduce a constraint satisfaction problem, CSP, which we have formulated in order to form this schedule and satisfy the needed constraints, form the complete schedule in no time, in the following report, we are going to discuss our design, how we have achieved a solution, and also an analysis of the complexity of this problem.

Introduction

One class of problems which we have studied in our Artificial Intelligence course is Constraint Satisfaction Problems, which has a goal for identifying the solution of a given problem in a way that is similar to the Depth First Search approach, but has some differences. Other than traversing a search tree which includes all of the possible states that could be found in the state space, it only explores nodes which do not violate any of the constraints which are property of the goal state, thus it cuts a great number of nodes which would be unnecessary to explore, a correct solution which is the goal state is achieved when we have an assignment for all of the problems' variables all of which satisfy the problem constraints.

The scheduling problem is a perfect example of an assignment problem which needs to be solved given a set of constraints, which are mainly in order to avoid conflicts in schedules for students in the same batch / major, that is why we saw potential in applying CSP evaluation and backtracking techniques to reduce the amount of efforts it would require to formulate a university schedule manually.

PROBLEM DESCRIPTION

While designing the problem, we tried to estimate the variables and domains in order to be as close as possible to our actual situation at university.

However, there were a few assumptions that we needed to make which are :

- Any room available is suitable to be used for the purpose of delivering a lecture or a lab
- A student from a certain academic year / major does not take courses from another year / major, meaning that conflicts only are a problem for each specific batch in a major
- Any room is available for teaching during all of the week days, all of the working hours

The table below shows the specific parameters which define our main case problem.

No of academic years	5
No of majors	9
Courses / major / academic year	6 (maximum)
1 course	1 Lecture + 1 Lab
Durations	- Lecture : 2 hours - Lab : 3 hours
No of teaching rooms available	30 (rough estimation of the number of rooms at our university)
No of days / week	5
No of hours / day	8

From the parameters which are specified, we can deduce that the available number of hours available for teaching at the university (the main resource) is equal to

$$slots = Rooms * Days * Hours = 1200$$

The required slots which are needed in the goal state

$$Majors * Years * Courses * Hours = 1140\ slots$$

including the foundation year.

This simple calculation indicates that the maximum number of courses offered by the university cannot exceed that we have specified above, or else we would need to add extra rooms to offer more slot times, that is in fact asserted in our code to overcome our code getting stuck in the backtracking step.

ALGORITHM DESCRIPTION

Now there is a need to declare the definition of a slot in our code. In the project proposal, we have stated that in order to assign rooms and times to the university courses, we would need to define two CSPs, however, we have made a simple change which made us avoid having the need to solve it over two stages and instead solve over just one stage.

A slot is defined by a four digit number, the first two digits of the number identify the room number of this slot, third digit indicates the week day, and fourth indicates the hour, an example is shown below.

Slot name : 1231	- Room number : 12 - Day : Tuesday - Hour : 8.30-9.30 (first hour of the day)

This simple definition of the available slots, have combined both of resources into one, now this means one last thing, a lecture would require to be assigned two slots for two hours, which are constrained to be in the same room, and of course consecutive hours in the same day.

The goal state is a full assignment for all of the courses with the required number of slots depending on whether it is a lecture or a lab.

In the beginning the domain of all of the courses is the available list of slots defined.

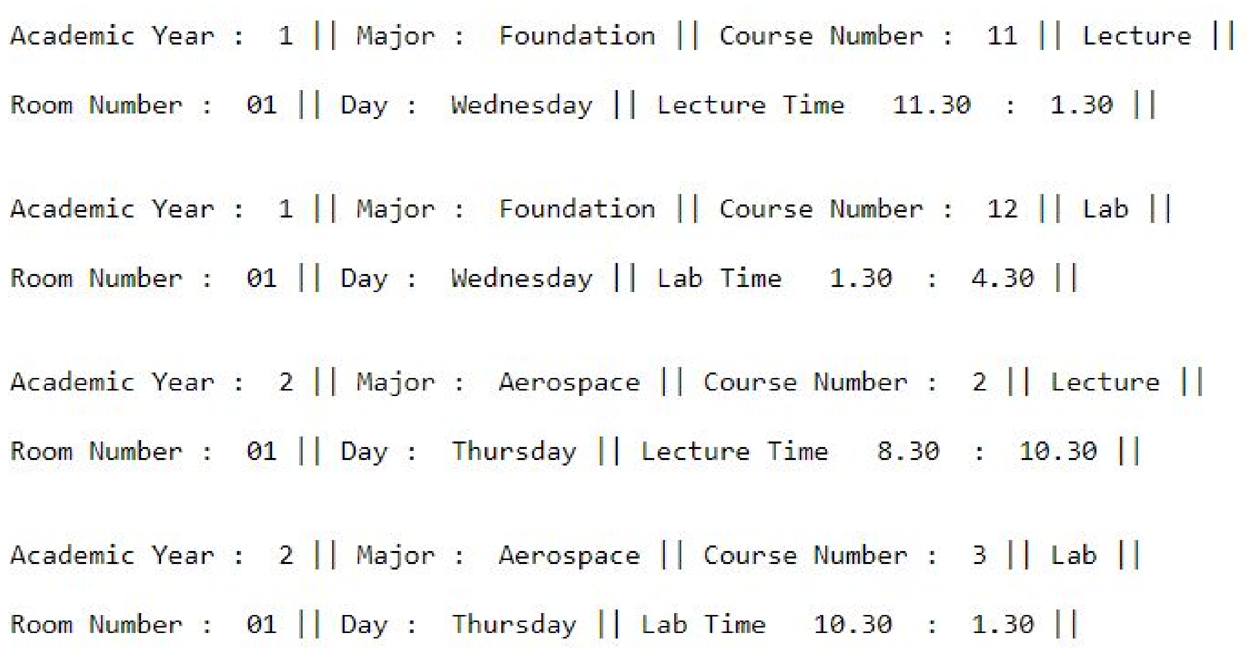
The following is a list of the constraints which we have provided to our problem :

- A slot can be used once and only once, no two lectures can share a room during the same day and hour
- A lecture has a duration of two hours, so it has to be assigned to two teaching hours in the same day and in the same room
- A lab has a duration of three hours, so it has to be assigned to three teaching hours in the same day and in the same room
- To avoid conflicts in the schedules of students, lectures and labs of all the students in the same academic year and same major cannot be concurrent.

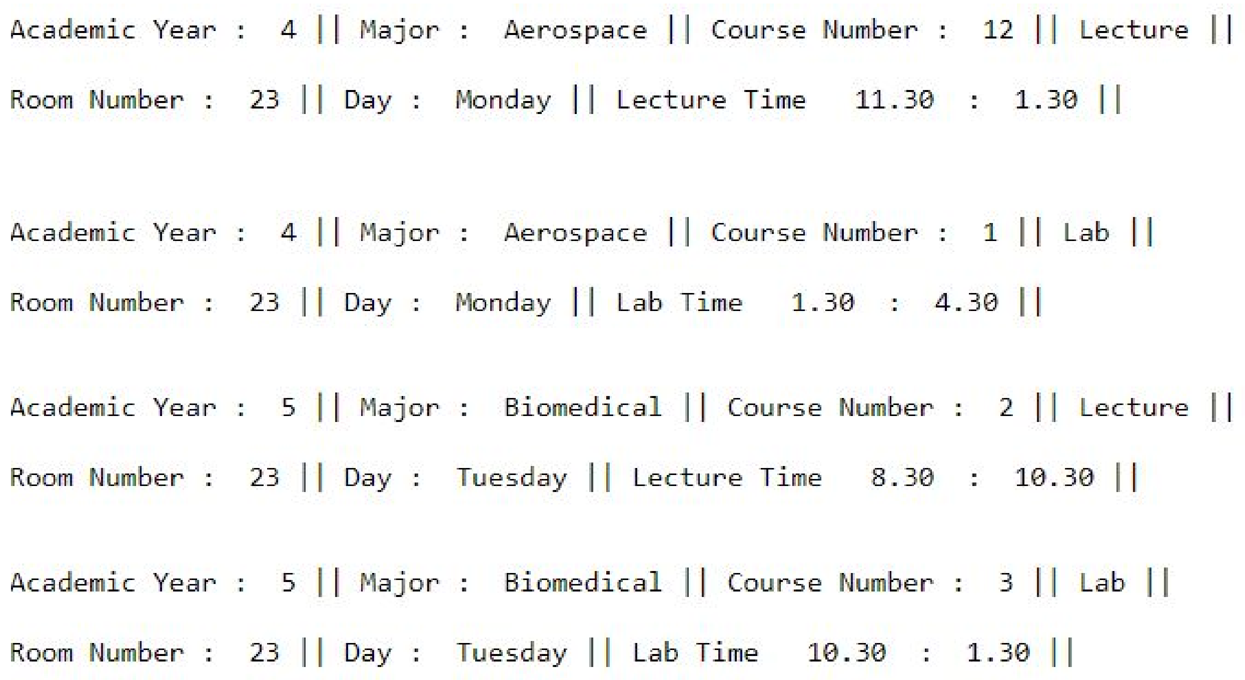
All of that is defined in a function that is called consistency, that acts as our evaluation function, and it checks if a new assignment of a variable to a certain value would be suitable to the variables that are already assigned or not, if all values in the given domain are not suitable, then a backtracking step is necessary because we will never reach a solution while traversing this path.

SAMPLE OUTPUT OF THE ALGORITHM

We created our own visualizers to be able to print the output in a readable format rather than the 4 digit representation, which consists of the year, the major, the course number, either being a lab or a lecture, its assigned room, the day, and the duration.



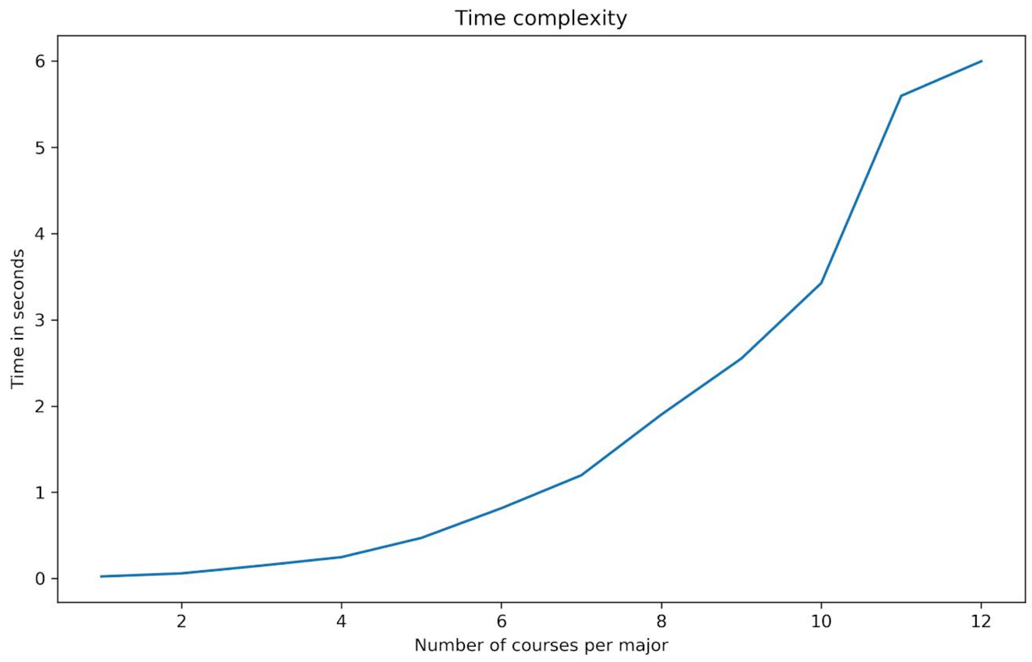
```
Academic Year :  1 || Major :  Foundation || Course Number :  11 || Lecture ||

Room Number :  01 || Day :  Wednesday || Lecture Time   11.30  :  1.30 ||

Academic Year :  1 || Major :  Foundation || Course Number :  12 || Lab ||

Room Number :  01 || Day :  Wednesday || Lab Time   1.30  :  4.30 ||

Academic Year :  2 || Major :  Aerospace || Course Number :  2 || Lecture ||

Room Number :  01 || Day :  Thursday || Lecture Time   8.30  :  10.30 ||

Academic Year :  2 || Major :  Aerospace || Course Number :  3 || Lab ||

Room Number :  01 || Day :  Thursday || Lab Time   10.30  :  1.30 ||
```



```
Academic Year :  4 || Major :  Aerospace || Course Number :  12 || Lecture ||
Room Number :  23 || Day :  Monday || Lecture Time   11.30  :  1.30 ||

Academic Year :  4 || Major :  Aerospace || Course Number :  1 || Lab ||
Room Number :  23 || Day :  Monday || Lab Time   1.30  :  4.30 ||

Academic Year :  5 || Major :  Biomedical || Course Number :  2 || Lecture ||
Room Number :  23 || Day :  Tuesday || Lecture Time   8.30  :  10.30 ||

Academic Year :  5 || Major :  Biomedical || Course Number :  3 || Lab ||
Room Number :  23 || Day :  Tuesday || Lab Time   10.30  :  1.30 ||
```

PERFORMANCE ANALYSIS

While CSPs are usually not easy to analyze mathematically due to their nature, we were able to estimate the time complexity of our solution by changing the number of courses per major and calculating the time needed to solve the problem. This was done for 12 times only, as more than that will not be applicable with the university current number of rooms. As expected, our solution is tending to be an exponentially increasing, therefore, we can estimate the time complexity as $O(2^n)$, where n is the number of courses per major. This is a rough estimation that can be not entirely true if we were able to solve the problem on a larger number of variables.



ALGORITHM LIMITATIONS

In the beginning of the problem definition, we have an assert condition, which checks if the number of required teaching hours in the semester is more than the available slots given the number of available rooms, that is in order to make sure that we have enough resources which are needed to cover this teaching time.

The problem is, the resources might be enough, but any combination of assignments to the variables would not satisfy the given constraints, and that is the main limitation of our code, that we can not check this beforehand, we need to run the algorithm, if a solution is returned, then it was enough, if no solutions were returned, then the constraints can not form a complete goal state.

CONCLUSION

In summary, we tackled a practical problem that universities face each semester, which is the registration of the students, using constraint satisfaction technique. While we made a few assumptions to loosen up the problem and make it more solvable, we still faced some problem with the time complexity of the problem. Therefore, with a larger number of variables, using more than one heuristic is inevitable.