CS255 Artificial Intelligence Coursework

Yalun Zhang
u1820624

I. Introduction

This report will detail the approaches taken to create a basic schedule that adheres to certain requirements, a similar schedule with lab sessions and a schedule of the minimal possible cost. For each of the three tasks, a solution is obtained using python3 for all of the example problems that are given, in a reasonable time frame. The design choices made are justified throughout with an evaluation of the solution following for all three parts of the programming task.

II. Task One

A. Approach

To complete this task, it was important to first identify the constraint satisfaction problem – the set of variables, domains and constraints, to allow for a solution to be reached where each variable can be assigned a value that satisfies all constraints. The variables are the modules which can be given values of possible tutors and must all be assigned to a slot in the timetable object. As mentioned in the code's preamble, there will always be 25 modules, one for each slot in the week, and a varying number of tutors.

The constraints are that:

- A tutor can only teach a module if the tutor's expertise topics are a subset of the module's topics, both of which are accessed through the supplied methods
- A tutor can only teach a maximum number of two different modules in a week
- A tutor cannot teach more than one module in a single day

Having considered this constraint satisfaction problem, it became clear that a backtracking algorithm would be most appropriate to arrive at a solution. This means the program would recursively solve the problem by building a solution one assignment at a time and if any one solution fails to satisfy the constraints, it is removed.

Furthermore, as given by the task description, this is a decision problem [1] where only a feasible solution is searched for, as there is no mention of searching for the most optimal solution. Despite this, it was still important to improve the search efficiency through general-purpose methods in order to gain large improvements in speed.

As a result, during the design stage of the process, a method to select the next variable to be assigned was devised. This was accomplished using the minimum remaining values heuristic [2], which in this CSP meant that the module with the fewest number of tutors that could teach it was chosen first. This is highly beneficial as the algorithm will pick the module most likely to cause the timetable to fail and assign it first, and therefore reduce the number of "backtracks" - which are caused by assignments that fail. To accomplish this, it was decided that a function called "generateOrder" would return a sorted list of modules, based on the number of tutors that could teach it in ascending order. Each element in this list contains the module object, the number of tutors that can teach it and a list of these tutors. Implementing this function required the use of the "issubset" method to check if a tutor's topics align with the module's and also the "sorted" method to arrange the list of tutors in ascending order based on the given key. The ordered list of modules is then used to populate the timetable object, with the tutor being blank, so that the provided "getSession" method can be utilised without any errors during the backtracking.

To implement the actual backtracking method, the base case is first established so that the recursion can end. For this task, the algorithm should terminate and return the list of assignments when there have been 25 successful assignments, meaning all of the slots in the timetable have been filled. The next variable (module) to be assigned is the element in the sorted list of variables at index "count", which is incremented each time there is a recursive call. The list of tutors that can teach the module created in the "generateOrder" function is iterated through, so if the algorithm backtracks, the next tutor in this list is chosen. For each slot in the timetable, if it is unassigned and the assignment satisfies the day constraint as well as the week constraint then the session is added to the timetable. If the most recent recursive call returns false, this means no valid assignment could be made and therefore the algorithm "backtracks" to the previous assignment by removing it from the timetable.

Although for this task there would always be a solution to the problem, if this was not the case then the algorithm should check if there is any module that cannot be taught by any tutor and then fail immediately.

B. Evaluation

For all problems one to eight, the solution is able to arrive at a solution immediately with no backtracking required. This occurs as a result of the sorting that occurs prior to the recursion starting and reflects the increased efficiency that the heuristics bring. The algorithm was also tested on additional problem sets to ensure the solution was effective for a wide range of inputs. By utilising debugging print statements, the ordering of variables using the MRV heuristic was determined to be working correctly, and any backtracks were clearly identified. The approach taken on this task is very close to the optimal solution as adding further heuristics would have negligible impact on its running time. With the low size of the given input, the algorithm is extremely effective at finding a feasible solution.

III. Task Two

A. Approach

The approach to solving task two is very similar to the previous task, except for a few alterations to the constraints and also the variables. In this task, the number of variables increases from 25 to 50, with each module also having a lab session. The credit system also means that there are additional constraints to consider. Each week, a tutor can only teach a maximum of 4 credits, with modules counting as 2 credits and lab sessions counting as 1 credit. Furthermore, a tutor has a credit limit of 2 per day. Similar to the first task a tutor's topics must be a subset of the module's topics however, in this task a tutor can teach a lab session if any of their topics are one of the module's topics. As before, this is a decision problem so only a feasible solution is searched for with no mention of optimising the credit system, which is tackled in the third part.

With this new specification, it was important to first outline the design differences between the previous task and this one. With the addition of the lab sessions, the minimum remaining values heuristic would need to be applied to a list of 25 modules and 25 labs. This can be accomplished by iterating through the module list and first checking the number of tutors that can teach it if it is a module and then if it is a lab – which is achieved using the "issubset" method followed by the "any" method. This results in a list of the 50 sessions and the tutors that can teach each of these sessions, ordered by the number of tutors that can teach the respective module or lab. The actual backtrack algorithm is essentially identical to the previous task, except the session type of the variable is first checked to differentiate between a lab and a module. This differentiation is important when evaluating the constraints as they have different credit weightings. Besides this distinction, the algorithm operates in the same manner with a counter being incremented after each recursive call and the backtracking remains the same.

In order to make the code more legible, an additional method named "evaluateConstraints" was created which returns the number of credits a given tutor teaches on a given day and also in the whole week. This is vital when checking the constraints and allows for more modularised coding.

B. Evaluation

As before, the algorithm creates solutions for all eight of the example problem sets and has a near instant run time. The minimum remaining values heuristic is applied to all 25 of the labs and modules effectively, resulting in the correct ordering of sessions. Although more heuristics can be utilised, the difference would be negligible hence these are left for the third part when the runtime differences will be greater. This algorithm was also tested on additional problem sets and the solution was equally effective. Due to the nature of the example problem sets, all of the modules have fewer tutors that can teach them than the labs however, even for problems where this is not the case the algorithm still functions. This is due to the fact that all of the modules and labs are sorted in one list.

IV. Task Three

A. Approach

To complete this task, an adaptation of the previous task was required as this is an optimisation backtracking problem where the best solution is searched for. Building upon the credit system created in the previous task with additional costs, this task needs to produce a schedule of the minimal possible cost - which can be calculated by hand as £10,050. The costing system essentially adds soft constraints to the CSP, as the algorithm should always prefer to assign one tutor to two modules on consecutive days and similarly, one tutor to four modules on two separate days.

With these soft constraints in mind, it was important to design a method that would calculate the cost of making an assignment so that the backtracking algorithm can prioritise the assignments with the lowest cost. To accomplish this, two methods were created. The first takes a module or lab with the list of tutors that can teach it and returns a list of assignments sorted by the cost. The second method is the one that actually calculates the cost, being given the position in the timetable, the tutor and the module or lab.

To assist with the cost calculation, a slot heuristic was developed which would calculate the number of free slots on the given day so that if the cost of assignments are equal, they are sorted by this value. This ensures that the algorithm would prioritise splitting 4 labs across 2 days as this is required for the optimal solution – 4 labs on 1 day would reduce the number of tutors that can teach 4 labs with the maximum discount.

Another heuristic that was added was the tutor heuristic which is used when ordering the variables. It calculates how many modules a tutor can teach, so that the tutors are sorted such that tutors with fewer teachable modules are assigned first. This reduces the amount of backtracking and allows for the optimal solution to be reached quicker.

Throughout this task, when values have been sorted with the heuristic and are still equal they are sorted randomly so that each iteration of the backtracking will be different, in an attempt to be closer to the optimal solution. In line with this, the module lists are also shuffled with each iteration.

Other changes made to the previous task's code include the addition of the assignment list which stores each assignment so that after the optimal solution is reached it can be added to the timetable. Additionally, the variables are ordered each time a recursive call is made so that tutors which have reached the credit limit and modules or labs that have already been assigned are removed from the pool. This ensures the efficiency and run time of the solution is optimised.

B. Evaluation

The solution arrives at the most optimal solution for all eight of the provided example problems within the reasonable time frame. It is possible for the algorithm to reach the optimal solution upon the first iteration of the backtracking algorithm, although this is unlikely. For problems 6 and 8, the algorithm can take up to 1 minute which occurs due to the number of backtracks. This could be reduced by adding further heuristics which prevent the need for numerous backtracks when there are limited options for valid timetables. As all eight provided examples had an optimal solution of 10050, the algorithm caters for that, however, it is also adapted to run the while loop a certain number of times and then return the minimum cost assignment for cases when the optimal solution is greater.

CONCLUSION

In conclusion, all parts of the programming task were approached in a sensible manner with sufficient planning before the development stage. The resulting solutions fit the given specification and are well within the reasonable time frame. Tasks one and two utilise the minimum remaining values heuristic to order the modules prior to any recursive calls. Task three utilises many different techniques in order to prioritise assignments of the lowest cost and also checks forward to ensure that any assignments will allow for the optimal assignments in the future as well.

REFERENCES

[1] GeeksForGeeks – Backtracking Introduction. Available at: https://www.geeksforgeeks.org/backtracking-introduction/ [Accessed: 10/01/20]

[2] Griffiths. N. CS255 Artificial Intelligence: Constraint Satisfaction Problems – Minimum Remaining Values (Slide 16)