Lab Assignment 2

Github link -

Francisco Montañés – 100406009 – 100406009@alumnos.uc3m.es

Miguel Stampa – 100455376 – 100455376@alumnos.uc3m.es

Group 89

Heuristics and Optimization

UC3M

0. Table of Contents

1. Introduction

In this report we have described the modelling of the two parts desired: validation with Python Constraint and Planning with Heuristic Search. First, we will talk about part 1 explaining the model presented and implementation in Python, decisions made and how it works. Then, about the second part with heuristic search.

In those sections, we have analyzed the results obtained, as asked for, discussed problems encountered and discussed several problem situations with some extreme cases to show the way our model deals with them. Then, conclusions and references to documents and pages that were consulted is shown.

As it will be seen in the explanations, we have decided to give an important weight to show in deep the decisions taken by commenting them and, slightly, went over the code. Besides, we wanted to give different examples to prove our code works, together with extreme cases and how those cases could be solved.

We had to revise several theoretical concepts and exercises done in class in order to gain the knowledge needed to solve the tasks and work faster. Specially those regarding Python constraints and its syntax, and heuristic ones.

2. Part 1

Problem modeling

The constraint satisfaction problem consists of a finite set of values X, defined over domains D and a set of constraints C, and it is modelled as follows:

· $X = \{x_i: (id, Y,\ T, M,\ idsibling, S_{Yj})\}\ where\ their\ characteristics\ are$:

$$id = \{1, \dots n\}, year\ Y = \{1,2\}, troublesome\ T = \{X, C\}, mobility\ M = \{X, R\}\ and\ idsibling = \{1, \dots n\}, where\ n\ is\ the\ number\ of\ students\ given, n \leq 32$$

$The\ domain\ D\ of\ every\ variable\ are\ the\ seats\ available, from\ 1\ to\ 32\ (in\ rows\ of\ 4),$
$but\ we\ already\ constrained\ it\ from\ the\ start\ with\ the\ constraint\ about\ reduced\ mobility$
$needing\ blue\ seats\ and\ nonreduced\ having\ all\ available\ that\ we\ will\ talk\ about\ now$

· Constraints are as follows:

- Only one seat assigned to each student: $\forall k: |S_{Yk}| = 1$

- Seats for students with reduced mobility are special ones. $\left(M_j = \text{"R"}\right) \rightarrow (S_{1j} = \{1,2,3,4,13,14,15,16\})$ and $(M_k = \text{R}) \rightarrow (S_{2j} = \{17,18,19,20\})$ or: $\forall M_j = \text{"R"}: S_{1j} = \{1,2,3,4,13,14,15,16\}\ or\ S_{2j} = \{17,18,19,20\}$; and place next to them has to be kept empty: $\forall M_k = \text{"}R\text{"}, \forall k: \left(S_{Yj}\%4 == 1\right) \rightarrow (|S_{Yj} + 1| = 0)$
- A seat or people with reduced can be assigned to any student if not occupied. Already satisfied when defining the CSP at start.
- Troublesome students cannot seat closer to other troublesome students: neither closer to reduced mobility student. $\forall T_k = 1, \forall S_{Yk}: (S_{Yk}\%4 == 1) \rightarrow (T_{S_{Yk}-4} = 0, T_{S_{Yk}-3} = 0, T_{S_{Yk}+1} = 0, T_{S_{Yk}+4} = 0, T_{S_{Yk}+5} = 0)$, i.e., seats in front, diagonals, back, front and side cannot be assigned to troublesome students. Also: $\forall T_k = 1, \forall S_{Yk}: (S_{Yk}\%4 == 4) \rightarrow (T_{S_{Yk}-4} = 0, T_{S_{Yk}-3} = 0, T_{S_{Yk}+1} = 0, T_{S_{Yk}+3} = 0, T_{S_{Yk}+4} = 0\ )$, $\forall S_{Yk}: (S_{Yk}\%4 == 2\ or\ S_{Yk}\%4 == 3) \rightarrow (T_{S_{Yk}-5} = 0, T_{S_{Yk}-4} = 0, T_{S_{Yk}-3} = 0, T_{S_{Yk}-1} = 0, T_{S_{Yk}+1} = 0, T_{S_{Yk}+3} = 0, T_{S_{Yk}+4} = 0, T_{S_{Yk}+5} = 0\ )$; also cannot be seated near reduced mobility student:
- First year students must be assigned seats in front and second year in back, except if they are second year with a sibling in first year.
- If 2 students are siblings, they must be seated together except if one has reduced mobility. $\forall a, b (a > b): id_a == idsibling_b : (S_{2a} = S_{1b})\ and: (S_{2a}\%4 == 2) \rightarrow (S_{1b}\%4 = 1)\ and\ (S_{2a}\%4 == 3) \rightarrow (S_{1b}\%4 = 0)$ also satisfying that the one being older sits near aisle and both sited in front.

This CSP can be defined in several more ways. However, we have decided this one above which clearly shows the constraints not in the most efficient way. Note that when using () together with → we are defining if, then. Also, k gets values is {1,…,n}

Model implementation

We chose to write the code in Python although other programs could have been chosen such as C++. We took this decision because of its simple syntax in comparison to other languages because it is very easy to read and write which makes the program faster to be written and corrected. Besides, it was the one we had used in class which helps us in terms of understanding the code of some exercises that where used as examples of what we wanted.

We will comment in a summarized way what we have done in our program and then show and explain in more depth the tests we carried on. We have created several matrices to save the bus seats type arrangement, the seats that students take, their characteristics and some other using data also used for previous ones such as seats available for year 1 student and so on. To solve the imposed constraints, we developed functions for solving each of them. For the first constraint we just do 2 loops in which we check that the same seat has not been assigned to the more than 1 student. For the second one, we check that student has reduced mobility, then just check whether sit on left kept empty or on right by using mod operations (when sit on left empty, this is the odd seat; when sit used is odd, sit on right empty, mod2=0). Third one is solved, again, by using mod and loop over all the students, operations to keep the sits around troublesome students free of troublesome students or with reduced mobility. Fourth constraint is solved by looping over all the students and once checked the age assign to front if from first year or from second but with a sibling from first year; or to the back if from second year. Fifth constraint is solved in a very similar way. Sixth uses mod again as explained at start to sit the older one near the aisle. Not that here the constraints are explained in a different way to the one used in statement.

We tried to test several things to verify that our model works and suits all the constraints, we show it one a table:

Test	**Result**	**Constraints satisfied/explanation**	**Description**
01	4	Reduced mobility (only blue sites) and sit on back (2nd year)	1 student from 2nd year with r.m.
02	6	r.m., not together (free space next to r.m)	2 students from 2nd year with r.m. and one t.s.
21	48	No need to sit together the siblings, they are r.m.	2 students with r.m. and different years, siblings
22	8	Sit together the siblings, with older near aisle and in front part	2 students, siblings
23	16	2 brothers from same year can seat both on aisle and no need to move them to front	2 students from 2nd year, siblings
24	56	Sit satisfying r.m.	2 students, same year, one r.m.

03	0	Impossible to sit in 4 seats leaving one empty next to each r.m.	3 students (of 2nd year), r.m. and t.s.
04	36	Sit on sits for r.m. each in front or back depending on its year (r.m. constraints and t.s.)	4 students, each from a year, r.m. and t.s.
41	48	No need to seat siblings together since r.m., do not care about t.s. since they are r.m. and will never be together	All 4 r.m. being a pair siblings, all t.s.
06	0	Impossible to sit in 4 seats leaving one empty next to each r.m. in back side of front because trouble with front part of back (t.s. constraints)	6 students all r.m. and t.s. from years 2 (2) and 1(4)

Figure 1: Tests developed to prove program works

We have tried some more extreme changes such as having more troublesome students than the possible ones, same for students with reduced mobility. Both of them giving 0 as a solution.

3. Part 2

Problem modeling

The state space consists of all the nodes we will generate, each of which represents a unique order for the queue, dependent on the previous and current students inserted. Start state is an empty queue. Final state is unknown and will depend on number of students, where all must have an assigned seat with a minimum cost, giving the most optimal way to enter the bus. The actions we carry is to place students in the last free position inside the queue to reach a full queue. Goal state (final) will be known once our heuristic h(n)=0 has lower cost for every node.

Model implementation

First part of code reads and treats the data to make the information available more legible by saving it into matrices with different desired information on seat or characteristics from the students. Following, we create the Node class that has all the attributes we want on cost, queue, students' characteristics…Now, we reach the important part of the code, where the A* function is defined. Working in a similar way to the algorithm that we were taught in class. First creating the necessary arrays that we will use in the while loop, where we iterate until having no student left to enqueue and we have reached the node with the lowest cost. Following, we created a function to see which is the node with the lowest cost which just loops to check all node's cost and compare to get the one with minimum value. Similar is done for the lowest cost solution. Then, times to check if node's expansion is valid, i.e. if queue satisfies the constraints that we were told (no reduced mobility at end, no two reduced mobility students…). Then generate other queues that are valid by creating the allowed nodes and computing the heuristic too. Then there is a function which computes the cost of node expansion generated by checking which student is sitting when and corresponding cost. Finally, compute the heuristic values, minimum cost with simple functions depending on which of the 2 heuristics had been chosen. And print solutions.

Now we look at some of the tests that were carried to see how they work. Which are also on the calls file. As done in part 1 we use a table to show it more compactly and easily:

Test	**Explanation**
02	2 reduced mobility students, must fail (need one student to help them)
04	3 students, one t.s. works well
23	2 students (one t.s.), works well since no restriction
24	2 students one with r.m., works, first r.m. then the other that helps
31	3 students each of one type and works with r.m. at start and t.s. at end as asked for

Figure 2: Tests developed to prove part 2 works

4. Problems encountered

The greatest difficulty we encountered was at the start with modelling one of the constraints and it caused infinite solutions. From what we have done in class, this is harder, which makes it good to learn more. Nonetheless, it takes time, thinking and questioning to see concepts and solve desired tasks.

We had several problems in modelling part 1, from not being able to solve the aisle requirement for older sibling, from not being able to sit them together. Also, we had some problem with satisfying the requirement of printing the solutions in desired format.

Working with Python can be tricky specially if you have not yet studied it or worked with it. This makes it larger to get into working on the proper lab assignment. Similar to what it happens when you are not very familiar with terminal or Github.

Another problem we had to faced was related to testing cases with a great amount of students since the computer is not capable of generating all of them. To solve this one should, for instance, just print a small number of them.

5. Conclusion

Developing this project has given us more abilities in working with Python and to realize about its potential. It has given us the opportunity to see the power of this subject with a true-life application. So, at the end we manage to enjoy the subject which seems a very practical subject.

Some concepts from Python or heuristics had to be revised so that we could solve the several parts of the assignment as well as the tools we are not that familiar with although been using them for some weeks.

We have seen an increment in knowledge while developing this lab assignment. Once part 1 was modeled, we got confident in solving part 2 and writing the report through the process. We are now confident to solve more difficult problems and can now see that those solved in class and exams are quite easy in comparison.

At the end, it is both interesting and good to be able to do projects on real-life situation - continuing with the previous lab assignment- instead of projects or assignments of more abstract situations. All by seeing that what you are learning will be useful for tomorrow and that theoretical concepts are learned with a purpose.

Nonetheless, it is quite hard to do this practice when not have followed all the courses well because concepts have to be taken from several subjects (Operating Systems, Artificial Intelligence and Programming) which is also good because let us see how the areas of Informatic is interrelated and how to apply what we learn in one subject to the others.

6. References

- A list of commonly used Git commands – GitHub <<https://github.com/joshnh/Git-Commands>>
- Cómo navegar con archivos y carpetas en una terminal. <<https://terminalcheatsheet.com/es/guides/navigate-terminal>>
- Python. <<https://www.python.org/doc/>>
- Stack Overflow-Where Developers Learn, Share, & Build. <<https://stackoverflow.com>>