Lab Assignment 2: *Constraint Satisfaction and Heuristic Search*

Heuristics and Optimization.

Applied Mathematics and Computing, 2022-2023

Planning and Learning Group
Computer Science and Engineering Department

1 Goal

The goal of this lab assignment is for the student to learn how to model and solve constraint satisfaction and heuristic search problems.

2 Problem statement

The school management board is so pleased with the work we did when designing the bus routes they decided to continue our business relation. Now they want us to assign a specific seat for each student in the school buses. The arrangement of seats in said buses can bee seen in Table 1:

The table has a color scheme that describes the following properties of each space:

- The seats are numbered (from 1 to 32).
- The seats that are colored in blue are destined (preferably) to students with reduced mobility.
- Green colored spaces are aisles used by the students to walk.
- Red colored spaces represent both doors and the driver's seat.

The school has students with different sets characteristics and needs:

- First (younger) and second (older) year students.
- Students with reduced mobility.
- Troublesome students.
- Siblings.

29	25	21	17	door	13	9	5	1	driver
30	26	22	18		14	10	6	2	
AISLE									
31	27	23	19		15	11	7	3	
32	28	24	20	door	16	12	8	4	door

Tabla 1: Seat arrangement in a bus.

2.1 Part 1: Validation with Python Constraint

To cater the different needs of each student, the school board asks us to assign the seats in the bus complying with the following constraints:

1. Each student has one and only one seat assigned.
2. If there are students with reduced mobility, they will have to sit on seats designated for this purpose (marked in blue), and the seat right next to them has to be empty due to the special needs of these students. For example, if one of these students is assigned to seat number 1, then seat number 2 must remain free. If 2 is assigned to one of these students, then 1 must remain free.
3. A seat for people with reduced mobility can be assigned to any student if it is not occupied by a student with reduced mobility.
4. Troublesome students cannot sit close to other troublesome students or to any student with reduced mobility. For example, if a troublesome student is assigned to seat 23, no other troublesome student can be assigned to any of the seats 18, 19, 20, 22, 24, 26, 27 or 28. If a reduced mobility student is assigned to seats 17, then troublesome students cannot sit on seats 13, 14, 21 or 22.
5. First year students must use seats in the front of the bus (between the driver and the doors, in our example seats from 1 to 16). Second year students must sit in the back of the bus (behind the doors, in our example seats 17 to 32).
6. If two students are siblings they must be seated next to each other. For example, siblings could sit on seats 5 and 6 but not on 6 and 7 (since they would be separated by the aisle). If siblings are on different years, then an exception would be made to the previous rule (front for first year, back for second year): both would be placed in the front, and the older brother has to be assigned the seat closer to the aisle. If both brothers are troublesome they still have to sit together, but rules regarding other troublesome students still apply. If one sibling has reduced mobility they do not have to be seated together, but they still have to be seated in the same section (front or back, depending on the case and following previous rules).

The school board wants you to:

1. Model the problem as a CSP problem in a way that a solution to the problem represents a valid seat assignment for the students.
2. Using the library *python-constraint*, develop a program that encodes the previous model. Your code must encode the problem and find solutions.

Your program takes as input a file with the list of students that need to have seats assigned in the bus. You have to develop your own test cases in order to analyze the behavior of your program. Input files must follow the structure of this example:

```
1,1,C,X,3
2,2,X,X,0
3,2,X,X,1
4,2,C,R,0
5,1,X,X,0
6,1,X,X,0
7,2,C,X,0
8,2,X,R,0
...
```

Row structure:

```
Id, Year, Troublesome, Mobility, IdSibling
Id: Student ID (1, 2, ..., N)
Year: 1/2 (first/second year)
Troublesome: C/X (Troublesome/Not troublesome)
Mobility: R/X (Reduced/Not Reduced)
IdSibling: Id of their sibling (0 if none)
```

Your program must be executed on a terminal with the following command:

```
python CSPBusSeats.py <path students>
```

Where `path students` is the path to an input file listing students that need an assigned seat.

The program must create an output file in the same directory where the input file is located. The name of the output file will be `<students>.output` El programa debe generar un fichero de salida en el mismo directorio donde se encuentran los ficheros de entrada. El nombre del fichero de salida será de la forma `<students>.output`. For example, if the imput file is `students01`, the output file must be named `students01.output`.

The output file will have in the first line the number of solutions found, followed by at least one of the solutions (preferably more than one). An example of an output file would be:

```
Number of solutions: <n>
{'3XX': 11, '1CX': 12, '6XX': 15, '5XX': 16,
 '8XR': 18, '4CR': 20, '2XX': 31, '7CX': 32}
```

where the first element (formed by 3 characters) of each pair represents the ID of the student, if they are troublesome and if they have reduced mobility. The second element of the pair is the number of their assigned seat.

The calls needed to execute the designed test cases must be included as a bash script called `CSP-calls.sh`. An example of said script could be:

```
python CSPBusSeats.py ./CSP-tests/students1
python CSPBusSeats.py ./CSP-tests/students2
...
```

2.2 Part 2: Planning with Heuristic Search

Now that students already have an assigned seat, the only thing left to determine is the queue the students must form to enter the bus in order to minimize the time needed for this.

Initially, all students arrive at the bus stop but do not form a queue, since they do not know what place they should take in it. The idea is to determine the position of each student in the queue in a way that minimizes the total time needed to enter the bus, taking into account the following considerations:

- Students with reduced mobility take three times more time to enter the bus than other students.
- A student that is in the queue right behind a student with reduced mobility must help them get on the bus. Because of this, there cannot be two consecutive students with reduced mobility in the queue. Also, no student with reduced mobility can be the last person in the queue.
- When one student helps a student with reduced mobility to enter the bus, the time both students take to enter is exactly the time the student with reduced mobility takes, since they enter at the same time.
- A troublesome student will double the time the students in front and behind them take to enter the bus. Furthermore, if a troublesome student helps a student with reduced mobility enter the bus, the time taken by both of them would be double the time needed if the help was given by a non-troublesome student.
- A troublesome student will double the needed to enter the bus for ALL students that are behind him in the queue and have an assigned seat that is higher than his.

In this section you have to:

1. Model the problem of planning the queue to enter the bus as a search problem.
2. Implement the A* algorithm (in any programming language you choose) to solve the problem.

3. Design and implement heuristics functions (at least 2), that estimate the remaining cost of a solution so they can be used to guide heuristic search algorithms.

4. Design test cases and solve them with your implementation. These test cases must be designed taking into account the efficiency of your code.

5. Analyze and compare the performance of the algorithm depending with respect to the heuristics you implemented.

Your code has the following input parameters:

1. A parameter with the path to a file with the list of students *students* of the test case.

2. The input files with the list of students must have the extension ***.prob***. It must contain only one line that has the exact same format as the output solutions from the first part. For example:

```
{'3XX': 11, '1CX': 12, '6XX': 15, '5XX': 16,
 '8XR': 18, '4CR': 20, '2XX': 31, '7CX': 32}
```

3. A parameter that determines the heuristic function to be used. (***1*** for the first heuristic and (***2*** for the second one.

Test cases must be executed from a terminal with the following commandP:

```
python ASTARBusQ.py <path students> <heuristic>
```

Where `ASTARColaBus.py` is the script you implemented, `path students` is the path to the input file with the students and `heuristc` is the heuristic function identifier.

Calls to execute the test cases you design must be included in a script called `ASTAR-calls.sh`:

```
python ASTARColaBus.py ./ASTAR-tests/alumnos.prob 1
python ASTARColaBus.py ./ASTAR-tests/alumnos.prob 2
python ASTARColaBus.py ./ASTAR-tests/alumnos2.prob 1
python ASTARColaBus.py ./ASTAR-tests/alumnos2.prob 2
...
```

Where `./ASTAR-tests` is the path to a directory that contains the test cases. This allows the execution of multiple test cases in a single call.

Your program must produce two output files. They must be generated in the sime directory as the input files:

- A file with the solution that must contain in one line an initial configuration of the queue, where students are ordered according to their seat number, and in the next line the optimal configuration found with your implementation. For example:

```
INITIAL:  {'3XX': 11, '1CX': 12, '6XX': 15, '5XX': 16, ...}
FINAL:    {'6XX': 15, '3XX': 11, '1CX': 12, '7CX': 32, ...}
```

  Assuming the queue is ordered from left to right. The name of the file will be

  `<students>-<heuristic>.output`.

- A file with execution statistics that contains information about the search process, such as total time, solution cost, solution length and expanded nodes.

```
Total time: 145
Total cost: 54
Plan length: 27
Plan cost: 132
```

The file extension for the statistic file will be `.stat`: `<students>-<heuristic>.stat`.

3 Lab assignment development

To develop the lab assignment the use of github is mandatory. The repository must be created in the account of one and only one of the students of the group, who will add the other member as a collaborator, who must be able to clone, pull and push from the repository. The professor of the practical classes must also be added as a collaborator. The repository must contain the solutions to the problems and the report.

4 Requirements for the report

The report must be in PDF format and has a maximum of 15 pages. This includes the cover, table of contents and back cover. It should contain, at least:

1. A brief introduction explaining the contents of the document.
2. A description of the models, discussing the decisions carried out.
3. Analysis of the results.
4. Conclusions.

Important: Reports that are not in PDF format will lose 1 point in grading.

The report must not contain source code.

5 Grading

This lab assignment will be graded over 10 points. It is mandatory to perform at least the first part and to write the report. Points are distributed as follows:

1. Part 1 (4 points)
    - Problem modeling (1 point).
    - Model implementation (2 points).
    - Results analysis and test case design (1 point).
2. Part 2 (6 points)
    - Problem modeling (1 point).
    - Algorithm implementation (2 points).
    - Test case design and comparative analysis of the heuristic functions implemented (2 points).

When grading the proposed model, a correct model will make half of the points. To get all the points, the model must:

- Be properly formalized in the report.

- Be simple and concise.
- Be properly explained (it should be clear what is the purpose and meaning of each variable/constraint).
- All design decisions must be justified in the report.

When grading the implementation of the models, a correct model will make half of the points. To get all the points, the implementation must:

- Be exactly the model proposed in the report.
- Source code must be properly commented and organized. Variable names must be descriptive and comments must be added when needed.
- Test cases must be designed in a way that allows for the verification of the correctness of the implementation.

It will be checked that both members of the group performed push to the repository, to make sure both members contributed to the development.

6 Submission

The deadline for submitting the assignment is **December 15 at 23:55**.

Only one student from each group has to make submission on Aula Global:

- A single .zip named file.
- The file must be named `p2-NIA1-NIA2.zip`, where `NIA1` and `NIA2` are the last 6 digits of the NIA of each member of the group. Example: `p2-054000-671342.zip`.
- The report in PDF format must be included in the root of these directories and must be named NIA1-NIA2.pdf. The report in PDF format must also be submitted through Turnitin.

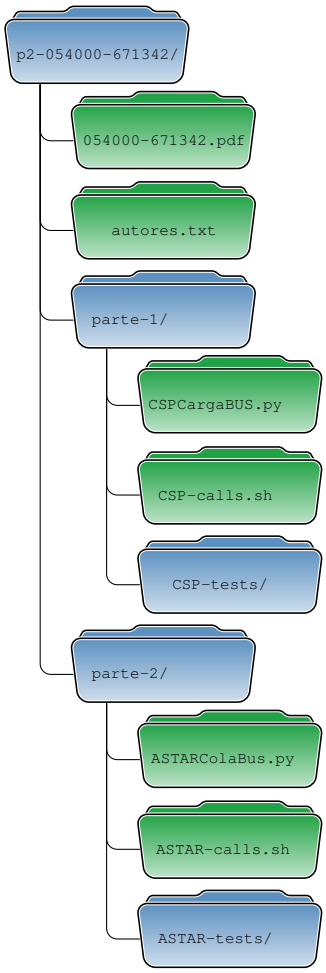
The extraction of the zip file must generate a single directory called `p2-NIA1-NIA2`. This directory must contain the report in PDF format submitted to Turnitin, and must be called `NIA1-NIA2.pdf`; a filed namedo *authors.txt* identifying each member of the group: one line for each member NIA Lastnames, Name. For example:

```
054000 Von Neumann, John
671342 Turing, Alan
```

The extraction of the zip file must also generate at least two directories called "`part-1`" and "`part-2`", that contain the files needed to execute each part of the lab assignment.

Important: Any submission made after the deadline or through different channels than the ones established will not be accepted. Ignoring other guidelines one way or another might result in the loss of up to 1 point in the final score..

Example of structure expected after extracting the zip file:


p2-054000-671342/
054000-671342.pdf
autores.txt
parte-1/
CSPCargaBUS.py
CSP-calls.sh
CSP-tests/
parte-2/
ASTARColaBus.py
ASTAR-calls.sh
ASTAR-tests/