uc3m

Practice: *Constraint Satisfaction and Heuristic Search.*
Heuristics and Optimization.
Academic Year 2025-2026

1 Objective

The objective of this practice is for the student to learn how to model and solve constraint satisfaction tasks, as well as to learn how to model and efficiently implement the resolution of heuristic search problems.

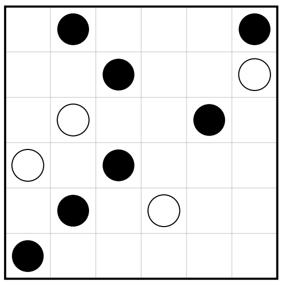
2 Problem Statement

The practice consists of three parts, detailed below, whose solution must be technically described according to the indications in section 3, and which have different weights and evaluations, as indicated in section 4. The practice *must* be done in pairs.

2.1 Part 1: Constraint Satisfaction

BINAIRO is a puzzle played on a square grid of dimensions $n \times n$, in which white and black discs must be placed, such that:

- No position must be left empty.
- The number of white and black discs in each row, and in each column, must be the same.
- It is not possible to place more than two discs of the same color consecutively in a row or column.


(a) Instance to be solved in 6×6

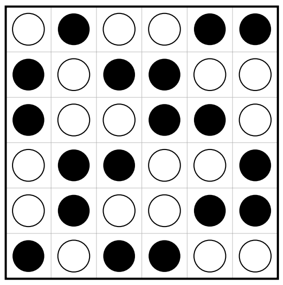

(b) Solution of the same instance

Figure 1: Example of a BINAIRO problem, on the left, along with a possible solution, shown on the right.

Initially, some cells may or may not already be occupied with white or black discs, as can be seen in the example in figure 1a, whose solution is shown in figure 1b.

You are asked to implement a Python 3.8 (or later) *script* called `parte-1.py` that solves any BINAIRO instance using `python-constraint`. The *script* must receive exactly two arguments: `input-file`, and `output-file`. Below are different use cases:

```
$ ./parte-1.py ejemplo.in ejemplo.out
$ ./parte-1.py /tmp/ejemplo.in ejemplo.out
$ ./parte-1.py ejemplo.in ../../datos/ejemplo.out
$ ./parte-1.py /tmp/ejemplo.in ../../datos/ejemplo.out
```

As can be seen, each file can be specified with an absolute or relative path, and if it does not contain any then the one where the *script* is located is assumed, which must:

- Read the input file indicated first, which must consist of n lines, each of which contains n symbols that can only be:
  - '`.`': indicates that the content of that cell is unknown.
  - '`X`': indicates that the content of that cell is a black disc.
  - '`O`': indicates that the content of that cell is a white disc.
- Solve the constraint satisfaction problem specified in the input file, and show the solution in the indicated output file, showing first the instance to be solved, and immediately after a solution to the problem.

  Additionally, the `parte-1.py` *script* must display on screen the instance to be solved, and then the number of solutions found.

Consider, for example, the following contents for the input file `ejemplo.in`:

```
.X...X
..X..O
.O..X.
O.X...
.X.O..
X.....
```

so that the execution of the *script* could produce an output like the following:

```
+---+---+---+---+---+---+
|   | X |   |   |   | X |
|   |   | X |   |   | O |
|   | O |   |   | X |   |
| O |   | X |   |   |   |
|   | X |   | O |   |   |
| X |   |   |   |   |   |
+---+---+---+---+---+---+
9 solutions found
```

and, additionally, it must write *one* solution to the indicated output file, `ejemplo.out`, preceded by the solved instance. A feasible solution could be:

```
+---+---+---+---+---+---+
|   | X |   |   |   | X |
|   |   | X |   |   | O |
|   | O |   |   | X |   |
| O |   | X |   |   |   |
|   | X |   | O |   |   |
| X |   |   |   |   |   |
+---+---+---+---+---+---+
+---+---+---+---+---+---+
| O | X | O | O | X | X |
| X | O | X | X | O | O |
| X | O | O | X | X | O |
| O | X | X | O | O | X |
| O | X | O | O | X | X |
| X | O | X | X | O | O |
+---+---+---+---+---+---+
```

where care has been taken to delimit each problem (the original instance, and its solution), to be able to recognize them both easily.

2.2 Part 2: Search Algorithms

The purpose of the second part of the practice is to find the *shortest* route between two points on one of the United States maps that are available in the 9th DIMACS Shortest-Path Challenge[1]. The specification of each graph consists of two files with suffixes `.gr` and `.co`:

- The first describes a *directed* graph indicating the edges between any two pairs of vertices uniquely identified by their index, with the distance between them on the Earth's surface in meters, and which are specified with lines that have the format:

  ```
  a <id1> <id2> <cost>
  ```

  being possible to ignore the rest of the lines.

- The second contains the longitude and latitude of each vertex on the Earth's surface multiplied by 10^6, in lines that have the format:

  ```
  v <id> <longitude> <latitude>
  ```

  being possible to ignore the rest of the lines.

For example, lines 8 and 9 of the file `USA-road-d.BAY.gr` are:

```
a 1 2 1988
a 2 1 1988
```

which indicates that from vertex `1` to `2` there is an edge with a distance of 1,988 meters and, likewise, it is possible to go from the second vertex to the first, traversing a distance also equal to 1,988 meters.

The coordinates file specifies, also in lines 8 and 9, that the longitude and latitude of each of these points is:

```
v 1 -121745853 37608914
v 2 -121745591 37610691
```

that is, vertex `1` is located at (37.608914, -121.745853), while vertex 2 is located at (37.610691, -121.745591) —where longitude and latitude have been intentionally inverted to express points on the Earth's surface according to international convention.

Figure 2 shows the two smallest graphs from the 9th DIMACS Shortest-Path Challenge with 264,346 and 321,270 vertices each, respectively.

You are asked to implement a Python 3.8 (or later) *script* called `parte-2.py` that solves any instance of the shortest path between any two points on any of the maps from the 9th DIMACS Shortest-Path Challenge. The *script* must receive exactly four arguments: `vertex-1 vertex-2 map-name`, and `output-file`. Below are different use cases:

```
$ ./parte-2.py 1 309 USA-road-d.BAY solucion
$ ./parte-2.py 1 309 /tmp/USA-road-d.BAY solucion
$ ./parte-2.py 1 309 USA-road-d.BAY ../../datos/solucion
$ ./parte-2.py 1 309 /tmp/USA-road-d.BAY ../../datos/solucion
```

[1]See `https://www.diag.uniroma1.it/~challenge9/download.shtml` in the columns *Distance graph* and *Coordinates*.

(a) Map of New York (`NY`)

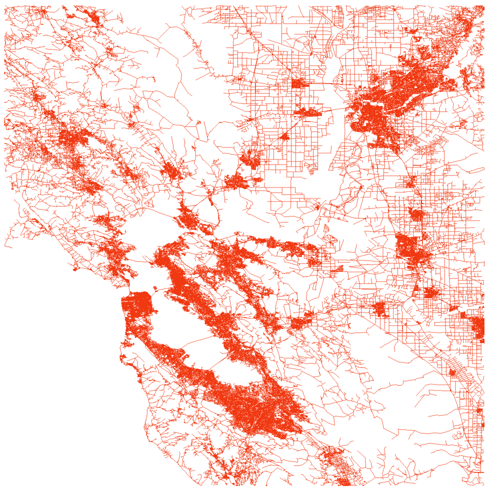

(b) Map of Bay Area (`BAY`)

Figure 2: Views of the two smallest graphs from the 9th DIMACS Shortest-Path Challenge

Note that the name `USA-road-d.BAY` uniquely identifies the graph files, `USA-road-d.BAY.gr`, and coordinates, `USA-road-d.BAY.co` which must be in the same directory. As can be seen, each file can be specified with an absolute or relative path, and if it does not contain any then the one where the *script* is located is assumed, which must:

- Read the graph and coordinates input files indicated in third place, which have the format indicated above.
- Solve the shortest path problem from the vertex indicated in first place, to the vertex indicated in second place, and write the solution to the file indicated in fourth place with the following format:

  ```
  <start> - ... - <vertex-i> - (edge cost <i,i+1>) - <vertex-i+1> - ... - <end>
  ```

  Additionally, the `parte-2.py` *script* must display on screen the following information: number of vertices processed from the coordinates file, number of edges processed from the graph file, cost of the optimal solution found and, then, the number of nodes expanded and the time spent solving the problem.

For example, the execution:

```
$ ./parte-2.py 1 309 USA-road-d.BAY solucion
```

could display the following information on screen:

```
# vertices: 321270
# edges   : 800172
Optimal solution found with cost 10216

Execution time: 0.15 seconds
# expansions: 4 (25.12 nodes/sec)
```

and the output file `solucion` would have the following contents:

```
1 - (1498) - 308 - (8718) - 309
```

so that the sum of the edge costs, 1498 + 8718, is exactly equal to the optimal solution cost found, 10216, shown in the output of the *script*.

To solve the problem you are asked to choose a search algorithm, and implement a heuristic that solves the problem *optimally* expanding fewer nodes than the same brute force algorithm, which you must also implement, and document in the results analysis —see section 2.3. The implementation of the *script* `parte-2.py` must contain the most efficient implementation possible using, for this, the best selection of algorithm, data structures and heuristic that is possible. It is required, in particular, that the implementation of this part contains at least the following files:

- `parte-2.py`: main *script* that solves the problem.
- `grafo.py`: definition of a class that serves to read the contents of a graph and represent them efficiently in memory.
- `abierta.py`: definition of a class with the implementation of the chosen open list, if your algorithm needs it.
- `cerrada.py`: definition of a class with the implementation of the chosen closed list, if your algorithm needs it.
- `algoritmo.py`: definition of a class that solves *optimally* the instance indicated in the parameters of the *script* `parte-2.py`.

2.3 Part 3: Results Analysis

In this section all the results obtained from each part must be analyzed, describing the solution obtained (checking that it meets the constraints of the statement) and analyzing how the execution time progresses as a function of the input problem variables. What explains the growth of execution time? What is the largest instance that your algorithm solves?

Analysis of problem complexity:

1. Regarding the first part, how many variables and constraints have you defined in the first part?
2. Regarding the second part, how does the size of the data structures defined in the second part grow? What reduction is possible in the number of expanded nodes and execution time in the second part thanks to the use of a heuristic? What algorithm have you chosen as the best to solve this problem?

Create different cases and demonstrate their resolution with the use of your *scripts*, varying the parameters, and explain how these modifications affect the difficulty of solving the resulting problem.

3 Guidelines for the Report

The report must be delivered in .pdf format and have a maximum of 15 pages in total, including the cover page, back cover and index, if any. **It must be a technical document**, instead of a user manual that contains, at least:

1. Brief introduction explaining the contents of the document.
2. Description of the models, arguing the decisions made, using for this a correct, clear and concise algebraic notation.
3. Analysis of the results, as indicated in section 2.3.

The report **must not include source code** in any case.

4 Evaluation

The evaluation of the practice will be done out of 10 points. For the practice to be evaluated, at least the first part of the practice must be completed:

1. Part 1 (3 points)
   - Problem modeling (1 point).
   - Model implementation (1 point).
   - Resolution and analysis of test cases (1 point).
2. Part 2 (7 points)
   - Problem modeling (2 points).
   - Algorithm implementation (3 points).
   - Resolution of different test cases and analysis of results (2 points).

In the evaluation of problem modeling, a correct model will account for half of the points. To obtain the rest of the points, the problem modeling must:

- Be correctly formalized in the report.
- Be, preferably, simple and concise.
- Be well explained (the purpose of each variable/constraint must be clear).
- Justify in the report all design decisions made.

In the evaluation of model implementation, a correct model will account for half of the points. To obtain the rest of the points, the problem implementation must:

- Fully correspond with the model proposed in the report.
- Deliver correctly organized and commented source code. Names must be descriptive. Comments should be added in cases where it is necessary to improve readability and understanding.
- Contain test cases that show diversity for the validation of the implementation.

Important: the implemented models must be correct. That is, they must work and obtain optimal solutions to each proposed problem. In no case will a grade higher than 1 point be obtained for a model that does not do so.

Generative AI: the use of Artificial Intelligence tools to carry out the practice is expressly prohibited, although it is possible to use them for the purpose of improving or reasoning about previously developed models.
In particular, the suspicion of fraudulent use of tools of this type, or any other consideration that leads to believe that some member of a team, or both, have not completed the practice entirely, is sufficient reason **to conduct an oral evaluation parallel to the grading of this practice**.

5 Submission

The deadline for submitting the practice is December 18, 2025, Thursday, at 23:55. This limit is fixed and will not be extended in any way. Take into account that the submission points for each reduced group have been configured to be able to overwrite previous versions.

Only one member of each pair of students must upload:

- A single `.zip` file through the 'Aula Global' submission point called "*Segunda práctica*".

  The file must be named `p2-NIA1-NIA2.zip`, where `NIA1` and `NIA2` are the last 6 digits of the NIA (filling with `0`s on the left if necessary) of each member of the pair.

  Example: `p2-054000-671342.zip`.

- The report must be uploaded separately through the Turnitin submission point of 'Aula Global' called "*Segunda práctica (sólo pdf)*".

  The report must be named `NIA1-NIA2.pdf` —after appropriately substituting the NIAs of each student as in the previous case.

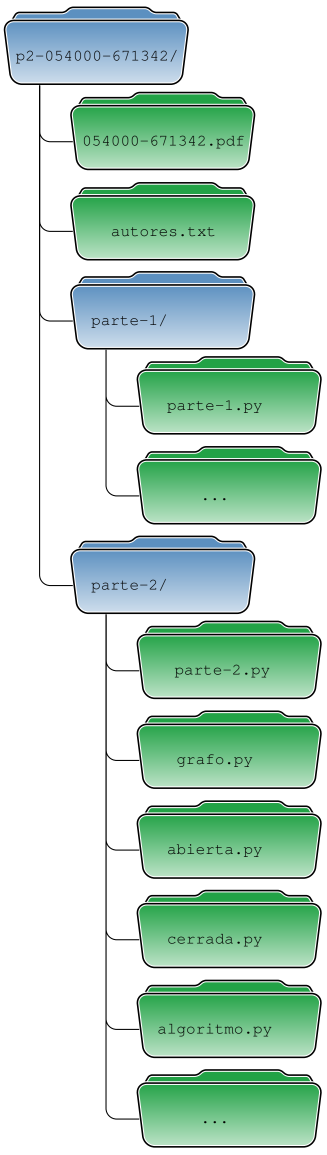
  Example: `054000-671342.pdf`.

The decompression of the `.zip` file described in the first point must produce a directory called `p2-NIA1-NIA2`, where `NIA1` and `NIA2` are the last 6 digits of each student's NIA filling with `0`s if necessary. This directory must contain: first, the same report delivered through the second link described above which must be named `NIA1-NIA2.pdf` —after appropriately substituting the NIAs of each student; second, a file called `autores.txt`, encoded in UTF-8, that identifies the members of each pair, with one line for each one with the following format: `NIA Lastname, Name`. For example:

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

Additionally, this directory must contain one directory for each completed part called "`parte-1`" and "`parte-2`". The solutions for each part must be included in their respective directories.

The following figure shows a possible distribution of files after decompressing the `.zip` file:



As can be seen, the directories “`parte-1`” and “`parte-2`” may contain other additional files (indicated in the figure with ellipsis marks). Valid examples are: additional Python files necessary for the execution of the *scripts*, or example data files of solved cases that are discussed in the report.

Important: not following the submission rules may result in a loss of up to 1 point in the final grade of the practice.