LINFO1361: Artificial Intelligence
Assignment 1: Constraint Programming to Solve Constraints Satisfaction Problems.

Alice Burlats, Achille Morenville, Harold Kiossou, Amaury Fierens, Eric Piette
February 2025

Guidelines

- This assignment is due on **Wednesday, 12th March 2025, 18h00**.
- *No delay* will be tolerated.
- *Document* your source code (at least the difficult or more technical parts of your programs). Python docstrings for important classes, methods and functions are also welcome.
- Copying code or answers from other groups (or from the internet) is strictly forbidden. Each source of inspiration must be clearly indicated.
- Source code shall be submitted on the online *INGInious* system. Only programs submitted via this procedure will be graded. No program sent by email will be accepted.
- Respect carefully the *specifications* given for your program (arguments, input/output format, etc.) as the program testing system is *fully automated*.

Requirements

For this assignment, you will use the python library **PyCSP3**. To install it, follow the instruction available here.

1 Understanding a CP model

The sudoku is a well-known puzzle game. The goal is to complete a partially filled 9×9 grid with digits between 1 and 9 such as:

- Each row contains all digits from 1 to 9.
- Each column contains all digits from 1 to 9.
- Each of the nine 3×3 subgrids contains all digits from 1 to 9.
- The values given in the initial grid need to be respected.

An example of a sudoku instance is shown in figure 1.

The file `sudoku.py` contains a model to solve the sudoku instance shown in figure 1. You can run it by using the following command in a terminal:

```
python3 sudoky.py
```

	6	2	5				7	
	8	5		6	7			9
					9			
	3					9	8	4
			1	3				
	2				5	6	1	
4	9		7					
		8	6	9			4	
7						1	9	

Figure 1: Sudoku instance

Task

The model contains several set of constraints. Your task for this exercise is to read and understand this model. You can then answer the MCQ available on inginious.

The file `alternative_sudoky.py` contains a model to solve a variant of the sudoku problem. Your task is to understand this model to identify which variant is modeled among the propositions on inginious.

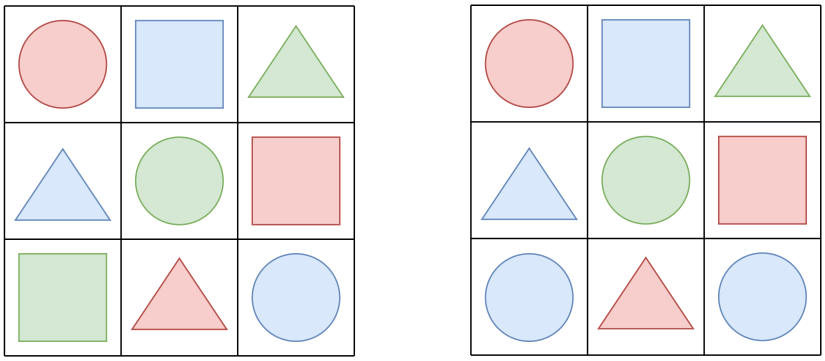
Grading

1. Answer correctly to the questions 1-4. (1pt) You can help yourself by looking at `https://www.pycsp.org/documentation/constraints/` to find the definition of the different constraints.
2. Answer correctly to the question 5. (0.5pt)
3. Find the right variant of the sudoku problem modeled in `alternative_sudoky.py` and answer correctly to question 6. (1pt)

   Warning ! The MCQ allows only one submission. So be careful when submitting your answers.

2 Tapestry Problem

The Tapestry Problem can be defined as follows. Given a tapestry represented as a $n \times n$ grid, you have to assign one shape and one color to each cell of the tapestry. For each cell you can choose among n colors and n shapes; the number of available shapes and colors is equal to the number of rows or columns in the tapestry. But the layout of the tapestry must



(A) (B)

Figure 2: An instance of the Tapestry Problem. (A) contains a valid solution and (B) an invalid solution.

follow several constraints:

1. Each shape must be present exactly once per row and column.
2. Each color must be present exactly once per row and column.
3. Each combination of shape and color must be unique in the entire tapestry. As an example, if a given cell contains a green circle, each other circle in the tapestry must have a different color and each other green shape can't be a circle.

Figures 2 below shows an example of a valid solution (A) and an invalid solution (B). The solution in (B) breaks each type of constraint:

- There are two circles in the first column. It breaks the constraint 1.
- There are two blue shapes in the first column and the last row. It breaks the constraint 2.
- There are two blue circles in the tapestry. It breaks the constraint 3.

Input

You are given an integer n corresponding to the dimensions of the tapestry and to the number of available shapes and colors. You are also given a $n \times n$ matrix **clues** representing the tapestry, where some initial values are already assigned. Each cell contains a tuple (s, c) where s is an integer between 0 and n representing the shape and c is an integer between 0 and n representing the color. If $s = 0$ or $c = 0$, no shape or color is assigned to the cell. A possible input could be:

$$\begin{bmatrix} (1,1) & (2,2) & (3,3) \\ (0,2) & (1,0) & (0,1) \\ (0,0) & (0,0) & (0,0) \end{bmatrix}$$

In this example, the cells in the first row are totally fixed. Your solution should return the same values for these cells. The second row contains cells where either the color or the

shape is unfixed. Your solution should respect the given color or shape while assigning the unfixed values. The third row contains cells where both the color and the shape are unfixed.

Output

You need to output a $n \times n$ matrix **solution** containing the shape and color of each cell. Considering the clues, **solution**$[i][j]$ should be equal to **clues**$[i][j]$ if **clues**$[i][j] \neq (0, 0)$. For example, the solution in figure 2.A should be represented as:

$$\begin{bmatrix} (1,1) & (2,2) & (3,3) \\ (3,2) & (1,3) & (2,1) \\ (2,3) & (3,1) & (1,2) \end{bmatrix}$$

Warning : Not all given input can lead to a valid solution. In the case of an unsatisfiable input, you should return None.

Task

Complete the file `tapestry.py` to solve the Tapestry Problem and submit your code here.

Grading

1. Your CP model returns a solution for each satisfiable input and None for each unsatisfiable input. (1pt)
2. The solutions returned by your CP model respect the clues given in the input and the constraints 1 and 2. (1pt)
3. The solutions returned by your CP model respect the constraint 3. (1pt)

3 Maniac Gardener Problem

Problem Description

A gardener wants to trim its hedges. The problem is that he is very strict on how they should look : Depending on the point of view a different number of them must be visible. More precisely, they are organized in a $n \times n$ grid. Each cell of the grid contains a hedge. Each hedge i can have a height h_i in $\{1, 2, \ldots, n\}$. If a hedge i is situated in front of a hedge j, i hides j iff $h_i \geq h_j$. For each row or column, the gardener has a precise number of hedges that can be visible. To have some diversity in his garden, he also wants that each hedge in a given row or column has a different height.

Figure 3 shows an instance of the Maniac Garder Problem. The garden is represented as a 4×4 grid of green cells, each cell being the representation of a hedge. The instructions are given by the dark cells. It reads as follows:

- Exactly 2 hedges must be visible from α.
- Exactly 2 hedges must be visible from σ.
- Exactly 4 hedges must be visible from β.
- etc...

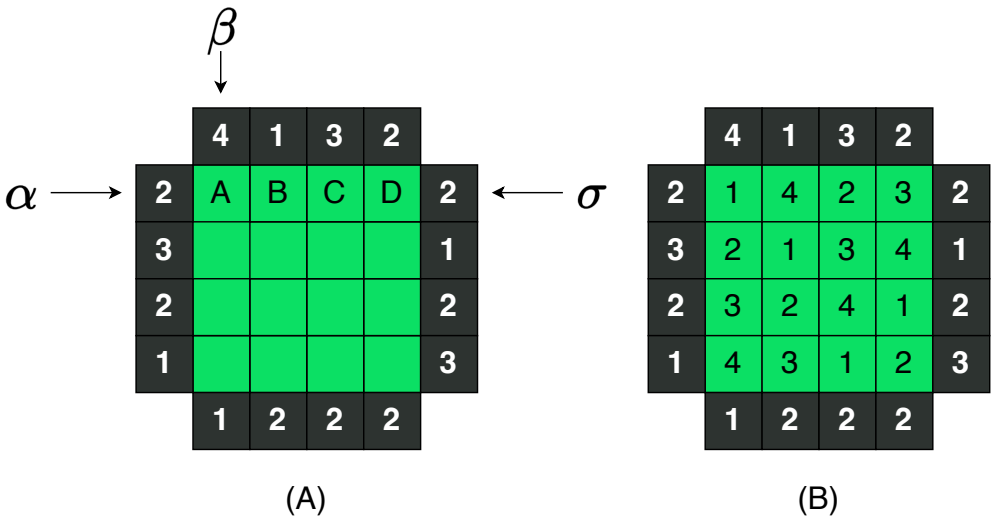


Figure 3: An instance of the Maniac Gardener Problem. (A) contains the original instance and (B) a valid solution to this instance.

Here are some precision on which hedges are considered visible and in front of/behind the other hedges. From the α point of view only the cells A, B, C and D can be visible. A is in front of B, C and D; B is in front of C and D but not A; etc.... From the σ point of view, it is the opposite: D is in front of A, B and C; C is in front of A and B; etc... From β, A can be visible, but neither B, C or D.

The figure 3.B shows the solution of this instance, where each number in a green cell represents the height of its corresponding hedge.

Input

You are given an integer n corresponding to the dimensions of the garden and to the maximal height of a hedge. You are also given a $4 \times n$ matrix *instructions*:

- *instructions*[0] corresponds to the top instructions.
- *instructions*[1] corresponds to the left instructions, *instructions*[1][0] corresponding to the instruction for the first row.
- *instructions*[2] corresponds to the right instructions, *instructions*[2][0] corresponding to the instruction for the first row.
- *instructions*[3] corresponds to the bottom instructions.

For example, the instructions for the instance in figure 3 are represented as:

$$\begin{bmatrix} 4 & 1 & 3 & 2 \\ 2 & 3 & 2 & 1 \\ 2 & 1 & 2 & 3 \\ 1 & 2 & 2 & 2 \end{bmatrix}$$

Output

You need to output a $n \times n$ matrix *solution* containing the height of each hedge. $solution[0][0]$ should correspond to the top left hedge. For example, the solution in figure 3. B should be represented as:

$$\begin{bmatrix} 1 & 4 & 2 & 3 \\ 2 & 1 & 3 & 4 \\ 3 & 2 & 4 & 1 \\ 4 & 3 & 1 & 2 \end{bmatrix}$$

Warning : Not all given input can lead to a valid solution. In the case of an unsatisfiable input, you should return None.

Restricted Problem

Because this problem is not trivial to model, we ask you to first solve a restricted version. In this version, you are given one instruction and a value for n. You need to design a CP model to find a row of size n such as the number of visible hedges (the cell at index 0 is in front of the other) is equal to the instruction, and such as each hedge has a different height. Figure 4 shows an instance of the restricted problem and its solution.

(A) (B)

Figure 4: An instance of the Restricted Maniac Gardener Problem.
(A) contains the original instance and (B) a valid solution to this instance.

Reification of a Constraint

In Constraint Programming, it is possible to constraints a variable to take a certain value if a condition is met. For example, consider a boolean variable y and an array of variables x.

$$y \Leftrightarrow AllDifferent(x) \tag{1}$$

This constraint enforces that y is true if and only if, all the value in x are different. Otherwise y is false. It is called **reification**.
Warning: Note that with this constraint, values in x are not forced to be different. The constraint is the **equivalence** between y and $AllDifferent(x)$. If a solution where each value in x is different **and** y is true is a valid solution, a solution where x contains duplicates **and** y is false is also a valid solution. A solution were x has no duplicates **and** y is false is not a valid solution.

For this problem you may need to use reification. You can find information on how to use it in with PyCSP3 at `https://www.pycsp.org/documentation/Combining`.

Task

You first need to complete the `restricted_gardener.py` file to solve the restricted version of the problem, and to submit your code here. Then you can complete the `gardener.py` file to solve the complete version of the problem, and submit your code here. The points are given as follows:

Grading

1. Your CP model is able to solve the restricted problem with one row and one direction only. (2pt)
2. Your CP model returns a solution for each satisfiable input and None for each unsatisfiable input. (1pt)
3. For the complete problem, your CP model returns a solution where each hedge has a unique height in a row or column. (1pt)
4. For the complete problem, your CP model returns a solution where the instructions are respected. (2pt)