LINFO1361: Artificial Intelligence
Assignment 4: Constraint Programming and Propositional Logic

Auguste Burlats, Eric Piette
April 2024

Guidelines

- This assignment is due on **Thursday 9 May, 6:00 pm**.
- *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*.
- The answer to questions must be given by filling in the latex template provided. The final file must be submitted on *gradescope*. No report sent by email will be accepted.
- Nothing must be modified in the template except your answer that you insert in the *answer* environments as well as your names and your group number. The names are provided through the command *students* while the command *group* is used for the group number. The dimensions of *answer* fields *must not* be modified either. For the tables, put your answer between the "&" symbols. Any other changes to the file will *invalidate* your submission.
- To submit on gradescope, go to `https://gradescope.com` and click on the "log in" button. Then choose the "school credentials" option and search for *UCLouvain Username*. Log in with your global username and password. Find the course LINFO1361 and the Assignment 4, then submit your report. Only one member should submit the report and add the second as group member.
- Check this link if you have any trouble with group submission `https://help.gradescope.com/article/m5qz2xsnjy-student-add-group-members`

Deliverables

- The answers to all the questions have to be submitted in a report with the given format on Gradescope. Do not forget to put your group number on the front page.
- The following files are to be submitted on Inginious:
  - `amazons_cp.py`: which contains `cp_model.py` method, to submit on INGInious in the *Assignment4: N-amazons CP* task.
  - `amazons_sat.py`: which contains `get_expression` method, to submit on

INGInious in the *Assignment4: N-amazons SAT* task.

1 Description of the N-amazon Problem

For this assignment, you will work on a problem that you encountered before: the N-Amazons. In the following sections, you will solve it by using two different methods: Constraint Programming and Propositionnal Logic. As a reminder, here is the problem statement taken from assignment 3.

This problem is a variant of the N-Queens problem that is harder to solve than the original. In the N-Amazons problem, the Amazon is a piece that moves like a queen, but have two other moves possible that are extension of the classical knight move:

1. 3X2 move: the amazon can go 3 tiles in one direction and 2 tiles in another direction
2. 4x1 move: the amazon can go 4 tiles in one direction and 1 tile in another direction

Figure 1 shows a visual representation of all tiles attacked by the amazon piece on a 10x10 chessboard. Each tile attacked is in red. As you may notice, the amazon piece we describe in our problem makes it a lot harder to solve than the N-Queens problem.

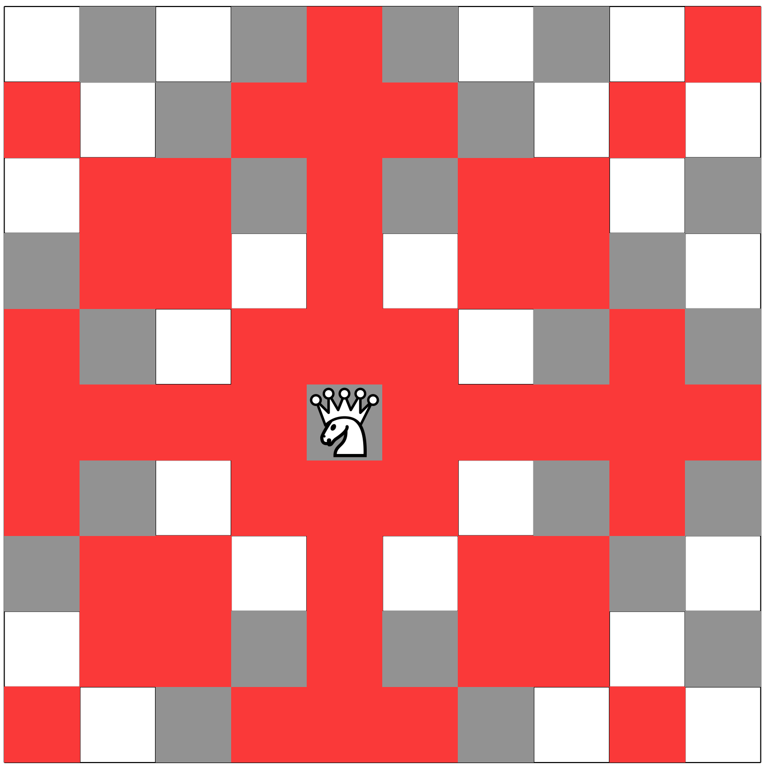
Figure 1

The directory *instances* contains instances for the N-amazon problem. In these instances, some amazons may have already been placed on the chessboard. In some of them, the already placed amazons make the problem unsolvable, there is no valid solution. The unsolvable instances are marked with the prefix *unsat_*.

The format is as follows:

Instance input format

$$\begin{array}{ll} n & p \\ a_i^0 & a_j^0 \\ a_i^1 & a_j^1 \\ \ldots & \\ a_i^{p-1} & a_j^{p-1} \end{array}$$

where n is the length and width of the chessboard and the number of amazons to place, p is the number of already placed amazons. The p following lines give their positions and are composed of two integers, respectively the column and row indexes. The chessboard origin $(0, 0)$ is at the top left corner.

2 Constraint Programming (7 pts)

For the constraint programming part of this assignment, you will use the python library PyCSP3. You first need to follow the instructions at `https://www.pycsp.org/documentation/installation` to install the library.

2.1 A simple model (1 pts)

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

Figure 2: Sudoku instance

The file `sudoku.py` contains a model to solve the sudoku instance shown in figure 2. The problem is modeled with a. 9×9 array of variables X where $X[i][j]$is a variable representing the cell (i, j). For each variable, its domain is $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$. There is also a 9×9 array of variables named *clue*, which contains the fixed values of the sudoku problem (if $clue[i][j] \neq 0$, then $X[i][j]$ needs to be equal to $clue[i][j]$). $X[0][0]$ is the cell located at the top left corner.

You can run it by using the following command in a terminal:

```
python3 sudoky.py
```

Questions

1. **(1 pt)** Identify the goal of each set of constraints. You can help yourself by looking at `https://www.pycsp.org/documentation/constraints/` to find the definition of the different constraints.

2.2 The N-amazons problem (6 pts)

Here you need to design a model to solve the N-amazon problem.

Questions

1. **(2 pt)** In constraint programming, there are several ways to model a problem. The first important decision is the used variables. Find at least two different sets of variables to model the N-amazon problem. For each, describe the variables that you will use to model the N-amazon problem. For each variable, describe what they represent **in one sentence**. Give their domains. If you use arrays of variables, you can give one explanation for the array as a whole (and not for each of its elements), and you need to give its dimensions. Choose the most appropriate set of variables. Justify your choice.
2. **(2 pt)** Give the constraints that you will use to model the N-amazon problem. For each constraint, also describe what it enforces. Your model must take account of the already placed amazons.

The file `amazons_cp.py` contains a model template to solve the N-amazon problem. You need to complete the `cp_model(n:int, placed_amazons:list[(int, int)])` function to model the problem. It takes two arguments:

- *n*: the length/width of the chessboard and the number of amazons to place.
- *placed_amazons*: a list of tuples (x, y) representing the already placed amazons. A (2, 3) tuple means that there is an amazon at position (2, 3).

The function should return **two elements**:

- a string *status*: containing the status of the problem. It should be *"UNSAT"* if the instance is unsolvable and *"SAT"* otherwise.
- a $n \times n$ array *solution*: a 2D array representing the solution, where solution[i][j] is 1 if there is an amazon at position (i, j) and 0 otherwise. If the problem is unsolvable, *solution* should be *None*.

To run your model, enter the following command in a terminal:

```
python3 amazons_cp.py INSTANCE_FILE
```

where `INSTANCE_FILE` is the N-amazons instance file.

Questions

3. **(2 pt)** Modify the `amazons_cp.py` file to implement your model. Be careful to have the right format for your solution. Submit your code on INGInious inside the *Assignment4: N-amazons CP* task. Your program will be evaluated on 10 instances of which 5 are hidden. We expect you to solve all the instances. An unsatisfiable instance is considered as solved if the solver returns *"UNSAT"*.

3 Propositional Logic (8 pts)

For the Propositional Logic part of this assignment, you will use the solver MiniSAT. MiniSAT is a small and efficient SAT-solver. Pre-compiled binaries for Mac and Linux are given with this assignment. If you are on Windows, you can either use the Windows Subsystem for Linux (WSL[1]), or compile from the sources available at `http://minisat.se`. You can also use the machines in the computer labs. A quick user guide can be found at `http://www.dwheeler.com/essays/minisat-user-guide.html`, but that should not be needed if you use the given `minisat.py` module.

3.1 Models and Logical Connectives (1 pt)

Consider the vocabulary with four propositions A, B, C and D and the following sentences:

- $\neg(A \wedge B) \vee (\neg B \wedge C)$
- $(\neg A \vee B) \Rightarrow C$
- $(A \vee \neg B) \wedge (\neg B \Rightarrow \neg C) \wedge \neg(D \Rightarrow \neg A)$

Questions

1. **(1 pt)** For each sentence, give its number of valid interpretations i.e., the number of times the sentence is true (considering for each sentence **all the proposition variables** A, B, C and D).

3.2 A simple model (1 pt)

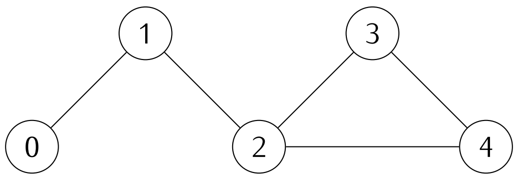

Figure 3: A graph coloring problem instance

The Graph Coloring Problem is a problem where you need to assign a color to each vertex of a graph such as:

- two neighbors (vertices connected by an edge) cannot have the same color.
- the number of used color must be inferior or equal.

The directory `graph_coloring_propositional_logic/` contains the model in propositional logic for one instance of the Graph Coloring Problem. The graph is represented in figure 3 and contains 5 nodes. The maximal number of available colors is 3. The different colors are represented by an integer among $\{0, 1, 2\}$. The problem is modeled with 5×3 variables, one variable for each pair (node, color): X_{ij} is true iff the node i is colored with the color j. Your goal is to identify the constraint enforced by each set of clauses.

The directory `graph_coloring_propositional_logic/` contains the following file:

- `solve_linux.py / solve_mac.py` is a python file used to solve the Graph Coloring Problem whether you machine is running on Linux or Mac.
- `graph_coloring.py` contains the definition of the clauses used to model the problem.

[1]For installation see `https://learn.microsoft.com/en-us/windows/wsl/install`

- `minisat.py` is a simple Python module to interact with MiniSAT.
- `clause.py` is a simple Python module to represent the clauses.
- `minisatLinux / minisatMac` is a precompiled MiniSAT binary that should run on the machines in the computer labs or your machine depending on the OS.

To solve the example of Graph Coloring Problem for instance on Linux machines, enter the following command in a terminal from the `graph_coloring_propositional_logic/` directory:

```
python3 solve_linux.py
```

Questions

1. **(1 pt)** Identify the goal of each set of clauses defined in `graph_coloring.py`.

3.3 N-Amazon problem (6 pts)

Your task is to model the N-Amazon problem with propositional logic. For this exercise, the set of literals is defined as follows: There are $n \times n$ variables: X_{ij} is *true* iff there is an amazon at position (i, j); *false* otherwise. The chessboard origin $(0, 0)$ is at the top left corner.

Questions

1. **(2 pts)** Explain how you can express this problem with propositional logic. For each sentence, give its meaning. Your model must take account of the already placed amazons.
2. **(2 pts)** Translate your model into Conjunctive Normal Form (CNF).

The `amazons_propositional_logic/` directory contains the following files:

- `solve_linux.py / solve_mac.py` is a python file used to solve the N-Amazons Problem whether you machine is running on Linux or Mac.
- `amazons_sat.py` is the skeleton of a Python module to formulate the problem into a CNF.
- `minisat.py` is a simple Python module to interact with MiniSAT.
- `clause.py` is a simple Python module to represent your clauses.
- `minisatLinux / minisatMac` is a precompiled MiniSAT binary that should run on the machines in the computer labs or your machine depending on the OS.

To solve the N-amazons Problem for instance on Linux machines, enter the following command in a terminal from the `amazons_propositional_logic/` directory:

```
python3 solve_linux.py INSTANCE_FILE
```

where `INSTANCE_FILE` is the N-amazons instance file.

Questions

3. **(2 pts)** Modify the function `get_expression(size)` in `amazon_sat.py` such that it outputs a list of clauses modeling the n-amazons problem for the given input. Submit your code on INGInious inside the *Assignment4: N-amazons SAT* task. The file `amazons_sat.py` is the *only* file that you need to modify to solve this problem. Your program will be evaluated on 10 instances of which 5 are hidden. We expect you to solve all the instances. An unsatisfiable instance is considered

as solved if the solver returns *"UNSAT"*.