Report LINFO1361: Assignment 4

Group N°13

Student1: Delsart Mathis

Student2: / (*Separate Duo*)

April 29, 2024

1 Constraint Programming (7 pts)

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

The first three constraints are ***'AllDifferent'*** constraints on **expressions**. This means that the expressions taken by the variables must all be different. The last one are constraints between two expressions taken by two variables.
Constraint #1: Ensure that all numbers in a **row** are **different** and **unique**. This constraint is applied to the ninth rows of the grid.
Constraint #2: Ensure that all numbers in a **column** are **different** and **unique**. This constraint is applied to the ninth columns of the grid.
Constraint #3: Ensure that all numbers in a **3x3 subgrid** are **different** and **unique**. This constraint is applied to the ninth subgrid of the grid.
Constraint #4: Ensure that all **initially placed numbers** on the grid remain the **same**.

2. 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 pt)**

The first choice is ***Individual Variables (One for Each Cell)***. In this case, the variables are $x_{i,j}$, where i and j represent the row and column indices respectively. $x_{i,j}$ represents whether an amazon is placed in cell (i, j) of the chessboard. The domain is therefore $\{0, 1\}$, where 0 represents no amazon and 1 represents an amazon. There are thus n^2 variables (n = size of chessboard) with 2 possible values for each variable. There are therefore 2^{n^2} states as a result.
The second choice is ***Array Variables (One for Each Column)***. The variables are x_i, where i represents the column index. x_i represents the row index where an amazon is placed in column i of the chessboard. The domain is therefore $\{-1, 0, 1, 2, ..., n-1\}$, where each value represents the row index of the amazon in that column. Here, -1 represents no amazon placed in the column. There are thus n variables with n possible values for each variable (n = size of chessboard). There are therefore n^n states as a result.
The **second model is superior for CSP** because it involves **fewer variables**, and the **domain** of each variable can be **easily reduced** compared to the first model. 2^{n^2} is **much worse than** n^n! This results in a **smaller search space**. Additionally, it allows for the definition of simpler and more effective constraints to ensure that each amazon is placed correctly and that there are no conflicts between the amazons. For instance, the constraint "only one amazon per row" can be easily expressed using the **AllDifferent** constraint on the array of variables (see next questions).

3. 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. **(2 pt)**

The ***first constraint*** **(**$\forall (i,j) \in$ **placed_amazons :** $x[j] == i$**)** ensures that the positions of the already placed amazons are respected in the final solution.
The ***second constraint*** **(**$\forall i \in \{0,1,\ldots,$ **size** $-1\} : x[i] \neq -1$**)** ensures that the total number of amazons placed on the chessboard is equal to the size of the chessboard. There must be exactly 'size' amazons placed. Here, -1 represents no amazon in the column.
The ***third constraint*** **(*AllDifferent***(x)**)** ensures that there is only one amazon per row on the chessboard. This constraint guarantees that no row contains more than one amazon.
The ***fourth constraint*** is implicit in the problem modeling and therefore does not need to be explicitly stated in the CSP. It ensures that there is only one amazon per column.
The ***fifth constraint*** **(**$\forall i,j \in \{0,1,\ldots,$ **size** $-1\} : i \neq j \Rightarrow |x[i] - x[j]| \neq |i-j|$**)** checks that there is only one amazon on each diagonal of the chessboard. This constraint verifies that the absolute difference between the positions of the amazons on the horizontal axis is different from the absolute difference between their positions on the vertical axis.
The ***sixth*** and ***seventh constraints*** ensure the validity of the 3x2 and 4x1 moves. This means that no amazon can be placed in a way that threatens another amazon on a movement of 3 squares horizontally (resp. vertically) and 2 squares vertically (resp. horizontally), or on a movement of 4 squares horizontally (resp. vertically) and 1 square vertically (resp. horizontally).
These constraints are defined as follows:
$(\forall i,j \in \{0,1,\ldots,$ **size** $-1\} : |i-j| = 2 \Rightarrow |x[i] - x[j]| \neq 3$
$(\forall i,j \in \{0,1,\ldots,$ **size** $-1\} : |i-j| = 3 \Rightarrow |x[i] - x[j]| \neq 2$
$(\forall i,j \in \{0,1,\ldots,$ **size** $-1\} : |i-j| = 1 \Rightarrow |x[i] - x[j]| \neq 4$
$(\forall i,j \in \{0,1,\ldots,$ **size** $-1\} : |i-j| = 4 \Rightarrow |x[i] - x[j]| \neq 1$
You can review the submitted code on Inginious to see the Python implementation.

4. Modify the `amazons_cp.py` file to implement your model. Be careful to have the right format for your solution. 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"*. **(2 pt)**

2 Propositional Logic (8 pts)

1. 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). **(1 pt)**

I reached these conclusions through logical deduction and then confirmed them using a Python script. The script iterated through all possible interpretations (16 in this case, due to the 4 variables) and tallied the number of valid interpretations, yielding the same result.
$\neg(A \wedge B) \vee (\neg B \wedge C)$: 12
$(\neg A \vee B) \Rightarrow C$: 10
$(A \vee \neg B) \wedge (\neg B \Rightarrow \neg C) \wedge \neg(D \Rightarrow \neg A)$: 3

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

Clauses 1: This clause ensures that **a node is colored with at least one color**. It is **true if at least one color is assigned to the node**, which is represented by positive literals for each possible color. This clause is applied to each node, ensuring that **at least one (1, 2, 3, ...) color is chosen** for each node.
Clauses 2: This clause **prevents a node from being colored with two different colors**. It is **true if at most one color is assigned to the node**, which is represented by negative literals for each pair of possible colors. This clause is applied to each node, ensuring that a node is assigned with **at most one color (0 or 1)**.
Clauses 3: This clause ensures that **two adjacent nodes are not colored with the same color**. It is **true if the color assigned to the adjacent nodes is not the same**, which is represented by negative literals for each possible color between each adjacent nodes. This clause is applied to each edge and for each color, thus avoiding color conflicts between adjacent nodes.

3. Explain how you can express the N-amazons problem with propositional logic. For each sentence, give its meaning. Your model must take account of the already placed amazons. **(2 pt)**

To express the N-amazons problem using propositional logic, we define the proposition:
X_{ij} := *there is an amazon in the square at row i and column j.*

1. **Already Placed Amazons**:
We assert the existence of already placed amazons by asserting the truth of the X_{ij} proposition for those specific tiles (i, j).
2. **At Least One Amazon per Row and Column**:
For each row i, we assert that there exists at least one tile (i, j) such that X_{ij} is true. Similarly, for each column j, we assert the existence of at least one tile (i, j) such that X_{ij} is true.
3. **No Two Amazons on the Same Row or Column**:
For each row i, we assert that for any pair of distinct tiles (i, j) and (i, k), if X_{ij} is true, then X_{ik} must be false, and vice versa. Similarly, for each column j, we assert the same condition for pairs of distinct tiles (i, j) and (k, j).
4. **At Most One Amazon per Diagonal**:
For each pair of cells where the absolute difference in their horizontal position matches the absolute difference in their vertical position, we establish a condition such that if one cell implies X_{ij} to be true, then the other must be false. This ensures that two amazons will not be on the same diagonal.
5. **At Most One Amazon per 3x2 Move and 4x1 Move**:
Similarly, For each pair of cells where the absolute difference in their horizontal position matches the value 3 (resp. 2, 4, 1) and the absolute difference in their vertical position matches the value 2 (resp. 3, 1, 4), we establish a condition such that if one cell implies X_{ij} to be true, then the other must be false. This ensures that two amazons will not be on two tiles that attack each other by the 3x2 Move or the 4x1 Move.

You can review the submitted code on Inginious to see the Python implementation.

4. Translate your model into Conjunctive Normal Form (CNF). **(2 pt)**

This is all the clauses to define the N-amazons problem with propositional logic. n represents the size of the chessboard minus 1.

1. $\bigwedge_{(i,j)\in placed\_amazons} X_{ij}$
2. $\left(\bigwedge_{i=0}^{n}\bigvee_{j=0}^{n}(X_{ij})\right) \wedge \left(\bigwedge_{j=0}^{n}\bigvee_{i=0}^{n}(X_{ij})\right)$
3. $\left(\bigwedge_{i=0}^{n}\bigwedge_{\substack{j=0\\k=0\\j\neq k}}^{n}(\neg X_{ij} \vee \neg X_{ik})\right) \wedge \left(\bigwedge_{j=0}^{n}\bigwedge_{\substack{i=0\\k=0\\i\neq k}}^{n}(\neg X_{ij} \vee \neg X_{kj})\right)$
4. $\left(\bigwedge_{i=n}^{0}\bigwedge_{j=0}^{n-i}\bigwedge_{k=j+1}^{n-i}(\neg X_{ij} \vee \neg X_{i+j,i+k})\right) \wedge \left(\bigwedge_{i=n}^{0}\bigwedge_{j=0}^{i}\bigwedge_{k=j+1}^{i}(\neg X_{ij} \vee \neg X_{i-j,i-k})\right)$
5. $\left(\bigwedge_{j=1}^{n}\bigwedge_{i=0}^{j}\bigwedge_{k=i+1}^{j}(\neg X_{ij} \vee \neg X_{i+j,i-k})\right) \wedge \left(\bigwedge_{j=1}^{n}\bigwedge_{i=0}^{j}\bigwedge_{k=i+1}^{j}(\neg X_{ij} \vee \neg X_{i-j,i+k})\right)$
6. $\bigwedge_{i=0}^{n}\bigwedge_{j=0}^{n}\left(\left(\bigwedge_{\substack{i+3\leq n\\j+2\leq n}}(\neg X_{ij} \vee \neg X_{i+3,j+2})\right) \wedge \left(\bigwedge_{\substack{i+3\leq n\\j-2\geq 0}}(\neg X_{ij} \vee \neg X_{i+3,j-2})\right) \wedge \left(\bigwedge_{\substack{i-3\geq 0\\j+2\leq n}}(\neg X_{ij} \vee \neg X_{i-3,j+2})\right) \wedge \left(\bigwedge_{\substack{i-3\geq 0\\j-2\geq 0}}(\neg X_{ij} \vee \neg X_{i-3,j-2})\right)\right)$
7. $\bigwedge_{i=0}^{n}\bigwedge_{j=0}^{n}\left(\left(\bigwedge_{\substack{i+2\leq n\\j+3\leq n}}(\neg X_{ij} \vee \neg X_{i+2,j+3})\right) \wedge \left(\bigwedge_{\substack{i+2\leq n\\j-3\geq 0}}(\neg X_{ij} \vee \neg X_{i+2,j-3})\right) \wedge \left(\bigwedge_{\substack{i-2\geq 0\\j+3\leq n}}(\neg X_{ij} \vee \neg X_{i-2,j+3})\right) \wedge \left(\bigwedge_{\substack{i-2\geq 0\\j-3\geq 0}}(\neg X_{ij} \vee \neg X_{i-2,j-3})\right)\right)$
8. $\bigwedge_{i=0}^{n}\bigwedge_{j=0}^{n}\left(\left(\bigwedge_{\substack{i+4\leq n\\j+1\leq n}}(\neg X_{ij} \vee \neg X_{i+4,j+1})\right) \wedge \left(\bigwedge_{\substack{i+4\leq n\\j-1\geq 0}}(\neg X_{ij} \vee \neg X_{i+4,j-1})\right) \wedge \left(\bigwedge_{\substack{i-4\geq 0\\j+1\leq n}}(\neg X_{ij} \vee \neg X_{i-4,j+1})\right) \wedge \left(\bigwedge_{\substack{i-4\geq 0\\j-1\geq 0}}(\neg X_{ij} \vee \neg X_{i-4,j-1})\right)\right)$
9. $\bigwedge_{i=0}^{n}\bigwedge_{j=0}^{n}\left(\left(\bigwedge_{\substack{i+1\leq n\\j+4\leq n}}(\neg X_{ij} \vee \neg X_{i+1,j+4})\right) \wedge \left(\bigwedge_{\substack{i+1\leq n\\j-4\geq 0}}(\neg X_{ij} \vee \neg X_{i+1,j-4})\right) \wedge \left(\bigwedge_{\substack{i-1\geq 0\\j+4\leq n}}(\neg X_{ij} \vee \neg X_{i-1,j+4})\right) \wedge \left(\bigwedge_{\substack{i-1\geq 0\\j-4\geq 0}}(\neg X_{ij} \vee \neg X_{i-1,j-4})\right)\right)$

The CNF is defined as $Clause\_1 \wedge Clause\_2 \wedge Clause\_3 \wedge Clause\_4 \wedge ... \wedge Clause\_9$.

5. 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. 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"*. **(2 pt)**