Report LINFO1361: Assignment 4

Group N°13

Student1: Delsart Mathis

Student2: / (*Separate Duo*)

April 22, 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 with 2 possible values for each variable. There are 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 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 queens placed on the chessboard is equal to the size of the chessboard. There must be exactly 'size' queens 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 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** $-1\} : |x[i]-x[j]| \neq 3$ **for** $|i-j| = 2$**)**
($\forall i,j \in \{0,1,\ldots,$ **size** $-1\} : |x[i]-x[j]| \neq 2$ **for** $|i-j| = 3$**)**
($\forall i,j \in \{0,1,\ldots,$ **size** $-1\} : |x[i]-x[j]| \neq 4$ **for** $|i-j| = 1$**)**
($\forall i,j \in \{0,1,\ldots,$ **size** $-1\} : |x[i]-x[j]| \neq 1$ **for** $|i-j| = 4$**)**
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)**

$\neg(A \wedge B) \vee (\neg B \wedge C)$: With 3 variables, there are $2^3 = 8$ possible interpretations. To satisfy this expression, at least one of the two parts must be true. For the left part, the only possibility is when both A and B are not true. For the right part, the only possibility is when B is false and C is true. Therefore, all interpretations are true except when both A and B are true. Thus, there are **6 true interpretations**.
$(\neg A \vee B) \Rightarrow C$: With 3 variables, there are $2^3 = 8$ possible interpretations. For this expression to be false, we need $True \Rightarrow False$. Thus, C must be false, and $(\neg A \vee B)$ must be true. This implies either A is false or B is true. Therefore, there are 4 cases leading to a false interpretation and hence **4 cases leading to a true interpretation**.
$(A \vee \neg B) \wedge (\neg B \Rightarrow \neg C) \wedge \neg(D \Rightarrow \neg A)$: For this expression, we observe that all three parts must be true. C must be set to false, and both A and D must be set to false. B has no restrictions. Thus, there are **2 true interpretations**.

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 a color contains at least one color**, and it is applied to all nodes to ensure that each node has a color.
Clauses #2: This clause **prevents a node from being colored with two different colors**. It is **always true except when a node is colored with two different colors**. It is also applied to all nodes to ensure that each node is assigned only one color.
Clauses #3: This clause ensures that **two adjacent nodes are not colored with the same color**. The clause **is always true except when two adjacent nodes are colored with the same color**. It is therefore applied to each edge, which represents the connection between two adjacent nodes.

3. Explain how you can express the N-amazons problem with first-order 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 first-order logic, we define predicates and quantifiers to represent the problem constraints. We define a predicate X_{ij} which is true if there is an amazon placed at tile (i, j), where i represents the row index and j represents the column index.
1. **Already Placed Amazons**: We assert the existence of already placed amazons by asserting the truth of the X_{ij} predicate for those specific tiles (i, j).
2. **At Least One Amazon per Row and Column**: We ensure that each row and each column contains at least one amazon by using existential quantifiers. 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**: We enforce the constraint that no two amazons share the same row or column by using universal quantifiers. 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**: We ensure that there is at most one amazon on each diagonal by defining predicates to represent the existence of amazons on each diagonal for each tile (i, j), and using existential and universal quantifiers to enforce constraints similar to those for rows and columns.
5. **At Most One Amazon per 3x2 Move and 4x1 Move**: Similarly, for each tile (i, j), we define predicates to represent the existence of amazons on moves corresponding to 3x2 and 4x1 configurations, and use quantifiers to enforce constraints to prevent multiple amazons on these moves.

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 first-order logic.

1. For each tile (i, j) $\in$ placed_amazons :
   (X_{ij})
2. For each row i :
   $(X_{i0} \vee X_{i1} \vee \ldots \vee X_{i(size-1)})$
   $(\neg X_{ij} \vee \neg X_{ik})$ for every pair of different squares j and k in the row.
3. For each column j :
   $(X_{0j} \vee X_{1j} \vee \ldots \vee X_{(size-1)j})$
   $(\neg X_{ij} \vee \neg X_{kj})$ for every pair of different squares i and k in the column.
4. For each tile (i, j) and $\forall k$ such that the condition is within the chessboard :
   $(\neg X_{ij} \vee \neg X_{(i+k)(j+k)})$ $(\neg X_{ij} \vee \neg X_{(i-k)(j+k)})$
   $(\neg X_{ij} \vee \neg X_{(i+k)(j-k)})$ $(\neg X_{ij} \vee \neg X_{(i-k)(j-k)})$
5. For each tile (i, j) and $\forall k$ such that the condition is within the chessboard :
   $(\neg X_{ij} \vee \neg X_{(i+3)(j+2)})$ $(\neg X_{ij} \vee \neg X_{(i+3)(j-2)})$
   $(\neg X_{ij} \vee \neg X_{(i-3)(j+2)})$ $(\neg X_{ij} \vee \neg X_{(i-3)(j-2)})$
   $(\neg X_{ij} \vee \neg X_{(i+2)(j+3)})$ $(\neg X_{ij} \vee \neg X_{(i+2)(j-3)})$
   $(\neg X_{ij} \vee \neg X_{(i-2)(j+3)})$ $(\neg X_{ij} \vee \neg X_{(i-2)(j-3)})$
6. For each tile (i, j) and $\forall k$ such that the condition is within the chessboard :
   $(\neg X_{ij} \vee \neg X_{(i+4)(j+1)})$ $(\neg X_{ij} \vee \neg X_{(i+4)(j-1)})$
   $(\neg X_{ij} \vee \neg X_{(i-4)(j+1)})$ $(\neg X_{ij} \vee \neg X_{(i-4)(j-1)})$
   $(\neg X_{ij} \vee \neg X_{(i+1)(j+4)})$ $(\neg X_{ij} \vee \neg X_{(i+1)(j-4)})$
   $(\neg X_{ij} \vee \neg X_{(i-1)(j+4)})$ $(\neg X_{ij} \vee \neg X_{(i-1)(j-4)})$

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

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)**