Experiment No-4

Problem Statement:

Implement a Solution for a Constraint Satisfaction problem using branch and bound and backtracking for n-queens problem or graph coloring problem

Theory:

Constraint Satisfaction Problems in Artificial Intelligence

We have seen so many techniques like Local search, Adversarial search to solve different problems. The objective of every problem-solving technique is one, i.e., to find a solution to reach the goal. Although, in adversarial search and local search, there were no constraints on the agents while solving the problems and reaching to its solutions.

In this section, we will discuss another type of problem-solving technique known as Constraint satisfaction technique. By the name, it is understood that constraint satisfaction means *solving a problem under certain constraints or rules.*

Constraint satisfaction is a technique where a problem is solved when its values satisfy certain constraints or rules of the problem. Such type of technique leads to a deeper understanding of the problem structure as well as its complexity.

Constraint satisfaction depends on three components, namely:

- **X:** It is a set of variables.
- **D:** It is a set of domains where the variables reside. There is a specific domain for each variable.
- **C:** It is a set of constraints which are followed by the set of variables.

In constraint satisfaction, domains are the spaces where the variables reside, following the problem specific constraints. These are the three main elements of a constraint satisfaction technique. The constraint value consists of a pair of **{scope, rel}**.

The **scope** is a tuple of variables which participate in the constraint and **rel** is a relation which includes a list of values which the variables can take to satisfy the constraints of the problem.

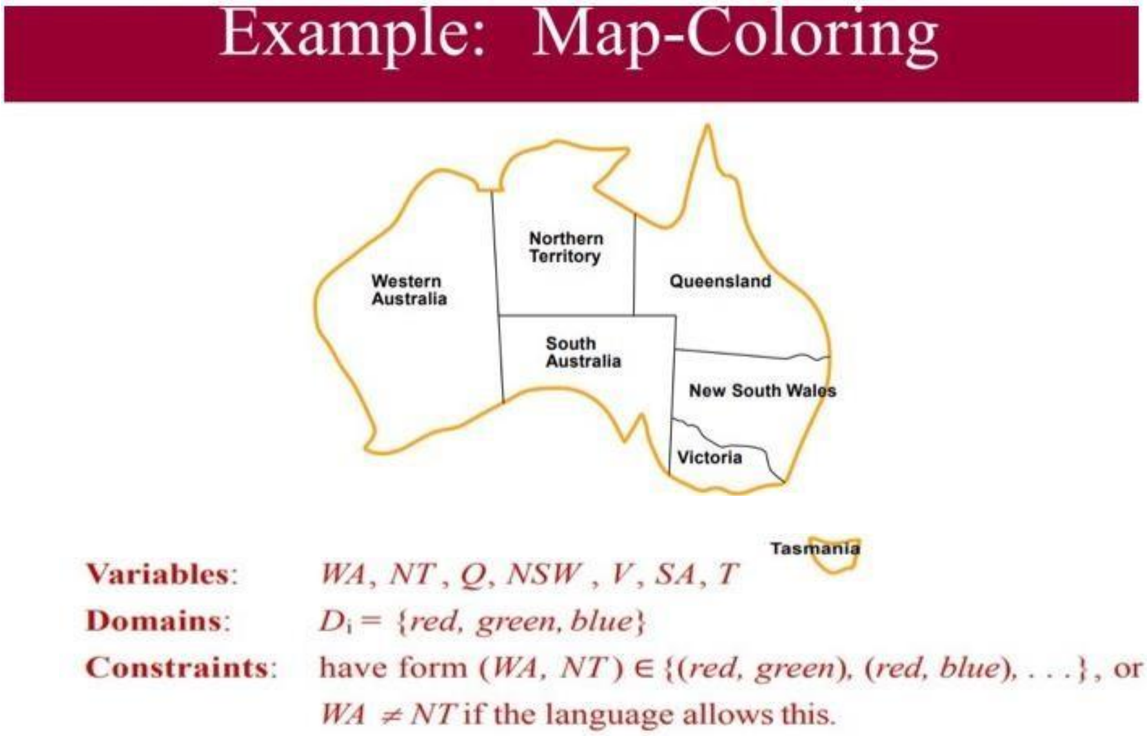
There are following two types of domains which are used by the variables:

- **Discrete Domain:** It is an infinite domain which can have one state for multiple variables. **For example,** a start state can be allocated infinite times for each variable.
- **Finite Domain:** It is a finite domain which can have continuous states describing one domain for one specific variable. It is also called a continuous domain.

Constraint Types in CSP

With respect to the variables, basically there are following types of constraints:

- **Unary Constraints:** It is the simplest type of constraints that restricts the value of a single variable.
- **Binary Constraints:** It is the constraint type which relates two variables. A value **x_2** will contain a value which lies between **x1** and **x3**.
- **Global Constraints:** It is the constraint type which involves an arbitrary number of variables.



Example: Map-Coloring

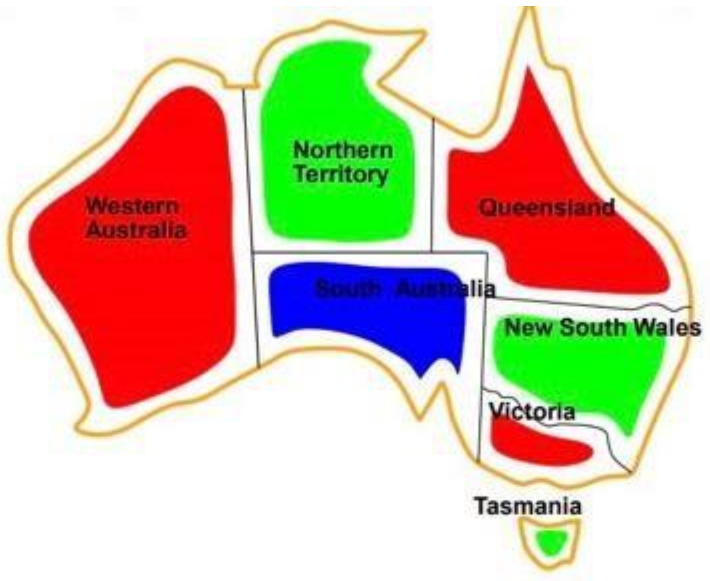
Variables: WA, NT, Q, NSW, V, SA, T

Domains: $D_i = \{red, green, blue\}$

Constraints: have form $(WA, NT) \in \{(red, green), (red, blue), \ldots\}$, or $WA \neq NT$ if the language allows this.

Example: Map-Coloring contd.

$\{SA \neq WA, SA \neq NT, SA \neq Q, SA \neq NSW, SA \neq V, WA \neq NT, NT \neq Q, Q \neq NSW, NSW \neq V\}$



One Solution: $\{WA = red, NT = green, Q = red,$
$NSW = green, V = red, SA = blue, T = green\}$

Applications of CSP:

- Map Coloring
- Scheduling the tasks
- Preparing Time Table
- Assignment
- Conflict Resolution
- Sudoku

8 Queens Problem using Branch and Bound

The N-Queens problem is a puzzle of placing exactly N queens on an NxN chessboard, such that no two queens can attack each other in thatconfiguration. Thus, no two queens can lie in the same row, column ordiagonal.

The branch and bound solution is somehow different, it generates a partial solution until it figures that there's no point going deeper as we would ultimately lead to a dead end.

In the backtracking approach, we maintain an 8x8 binary matrix for keeping track of safe cells (by eliminating the unsafe cells, those that are likely to be attacked) and update it each time we place a new queen. However, it required O(n^2) time to check safe cell and update the queen.

In the 8 queens problem, we ensure the following:

1. no two queens share a row
2. no two queens share a column
3. no two queens share the same left diagonal
4. no two queens share the same right diagonal

we already ensure that the queens do not share the same column by theway we fill out our auxiliary matrix (column by column). Hence, onlythe left out 3 conditions are left out to be satisfied.

Applying the branch and bound approach:
The branch and bound approach suggets that we create a partial solution and use it to ascertain whether we need to continue in a particular direction or not. For this problem, we create 3 arrays to checkfor conditions 1,3 and 4. The boolean arrays tell which rows and diagnols are already occupied. To achieve this, we need a numbering system to specify which queen is placed.

The indexes on these arrays would help us know which queen we are analysing.

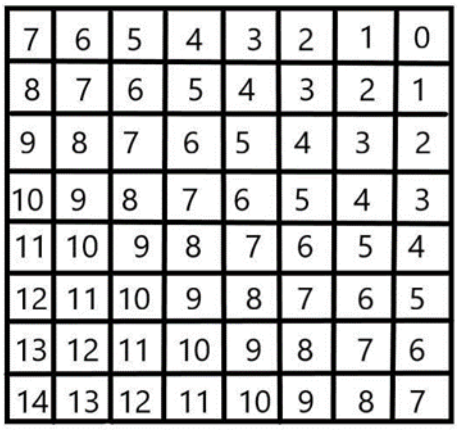
Preprocessing - create two NxN matrices, one for top-left to bottom- right diagnol, and other for top-right to bottom-left diagnol. We need to fill these in such a way that two queens sharing same top- left_bottom-right diagnol will have same value in slashDiagonal and two queens sharing same top-right bottom-left diagnol will have samevalue in backSlashDiagnol.

slashDiagnol(row)(col) = row + col

backSlashDiagnol(row)(col) = row - col + (N-

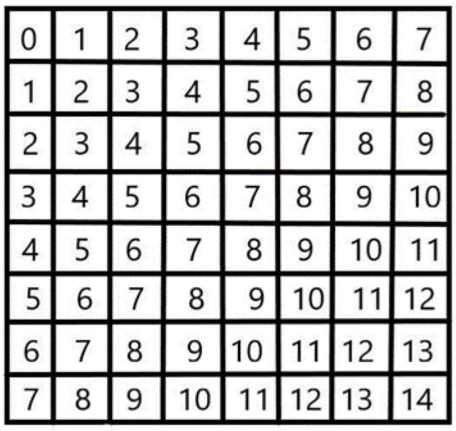
1) { N = 8 }

{ we added (N-1) as we do not need negative values in backSlashDiagnol }



7	6	5	4	3	2	1	0
8	7	6	5	4	3	2	1
9	8	7	6	5	4	3	2
10	9	8	7	6	5	4	3
11	10	9	8	7	6	5	4
12	11	10	9	8	7	6	5
13	12	11	10	9	8	7	6
14	13	12	11	10	9	8	7

slash diagnol[row][col] = row + col



0	1	2	3	4	5	6	7
1	2	3	4	5	6	7	8
2	3	4	5	6	7	8	9
3	4	5	6	7	8	9	10
4	5	6	7	8	9	10	11
5	6	7	8	9	10	11	12
6	7	8	9	10	11	12	13
7	8	9	10	11	12	13	14

backslash diagnol[row][col] = row-col+(N-1)

For placing a queen *i* on row *j*, check the following :

1. whether row 'j' is used or not
2. whether slashDiagnol 'i+j' is used or not
3. whether backSlashDiagnol 'i-j+7' is used or not

If the answer to any one of the following is true, we try another locationfor queen **i** on row **j**, mark the row and diagnols; and recur for queen **i+1**.

Conclusion:

We have studied Constraint Specification Problem with example.