

Lab Assignment 2

Subject: Artificial Intelligence

Guided by: Dr. Anuradha Yenikar

Student name: Neha Sawant

Experiment Name: Implement Constraint Satisfaction Problem for Sudoku

Objective:

- To solve the Sudoku puzzle using Constraint Satisfaction Problem (CSP) with backtracking.

Problem Statement:

Solve a 9x9 Sudoku puzzle where some cells are filled with numbers and others are empty (denoted by 0). The objective is to fill the grid such that each row, column, and 3x3 sub-grid contains all digits from 1 to 9 without repetition.

Algorithm:

1. Backtracking:

- Start by placing digits in empty cells.
- Check if placing the digit satisfies Sudoku constraints (row, column, and 3x3 sub-grid).
- If valid, place the number and continue solving the remaining puzzle recursively.
- If no solution is found, backtrack and try different digits.

Code Implementation (Sudoku Solver using CSP):

```

Run ... ← → AI_Labs
EightPuzzleBFS.java 3M_3C_problem.py sudoku.py X
AL_2. Constraint satisfaction problem > sudoku.py > solve
1
2 # Sudoku code simple
3 M = 9 # Size of the Sudoku grid (9x9)
4
5 # Function to print the Sudoku puzzle
6 def puzzle(a):
7     for i in range(M):
8         for j in range(M):
9             print(a[i][j], end=" ")
10            print() # New line after each row
11
12 # Check if a number can be placed in a given cell
13 def solve(grid, row, col, num):
14     # Check row and column for the number
15     for x in range(0):
16         if grid[row][x] == num or grid[x][col] == num:
17             return False
18     # Check the 3x3 sub-grid
19     startRow = row - row % 3
20     startCol = col - col % 3
21     for i in range(3):
22         for j in range(3):
23             if grid[i + startRow][j + startCol] == num:
24                 return False
25     return True # Safe to place the number
26
27 # Solve the Sudoku puzzle using backtracking
28 def Suduko(grid, row, col):
29     # Check if we reached the end of the grid
30     if (row == M - 1 and col == M):

```

```

... ← → AI_Labs
EightPuzzleBFS.java 3M_3C_problem.py sudoku.py X
AL_2. Constraint satisfaction problem > sudoku.py > solve
28 def Suduko(grid, row, col):
29     return True
30     # Move to the next row if at the end of the current row
31     if col == M:
32         row += 1
33         col = 0
34     # Skip filled cells
35     if grid[row][col] > 0:
36         return Suduko(grid, row, col + 1)
37
38     # Try numbers 1 to 9
39     for num in range(1, M + 1):
40         if solve(grid, row, col, num):
41             grid[row][col] = num # Place the number
42             if Suduko(grid, row, col + 1): # Recur for next cells
43                 return True
44             grid[row][col] = 0 # Reset if not successful
45     return False # Trigger backtracking
46
47 # Initial Sudoku grid with '0' representing empty cells
48 grid = [
49     [0, 7, 0, 0, 2, 0, 0, 0, 4, 6],
50     [0, 6, 0, 0, 0, 0, 0, 8, 9, 0],
51     [2, 0, 0, 8, 0, 0, 0, 7, 1, 5],
52     [0, 8, 4, 0, 9, 7, 0, 0, 0, 0],
53     [7, 1, 0, 0, 0, 0, 0, 0, 5, 9],
54     [0, 0, 0, 1, 3, 0, 4, 8, 0, 0],
55     [6, 9, 7, 0, 0, 2, 0, 0, 0, 8],
56     [0, 5, 8, 0, 0, 0, 0, 0, 6, 0],
57     [4, 3, 0, 0, 0, 0, 0, 0, 0, 0]
58 ]

```

```

Run ... ← → AI_Labs
EightPuzzleBFS.java 3M_3C_problem.py sudoku.py X
AL_2. Constraint satisfaction problem > sudoku.py > solve
60 ]
61
62 # Attempt to solve the Sudoku puzzle
63 if (Suduko(grid, 0, 0)):
64     puzzle(grid) # Print the solved puzzle
65 else:
66     print("Solution does not exist :)") # Indicate no solution found
67

```

Output Example:

```
PS C:\Users\nehas\Do
. Constraint satisfi
8 7 5 9 2 1 3 4 6
3 6 1 7 5 4 8 9 2
2 4 9 8 6 3 7 1 5
5 8 4 6 9 7 1 2 3
7 1 3 2 4 8 6 5 9
9 2 6 1 3 5 4 8 7
6 9 7 4 1 2 5 3 8
1 5 8 3 7 9 2 6 4
4 3 2 5 8 6 9 7 1
```

Conclusion:

From this assignment, I learned how to implement a Constraint Satisfaction Problem (CSP) to solve real-world puzzles like Sudoku. I gained a deeper understanding of how backtracking is used to explore possible solutions, ensuring that constraints (like unique digits in rows, columns, and sub-grids) are satisfied at each step. Additionally, I learned the importance of pruning in the solution space, which helps in optimizing the algorithm by reducing unnecessary computations.