CS512– Artificial Intelligence

Instructor : Shashi Shekhar Jha (shashi@iitrpr.ac.in)

Assignment - 3 | Due on 16/04/2020 2400 Hrs (50 Marks)

Submission Instructions:

All submission is through google classroom in one zip file. In case you face any trouble with the submission, please contact the TAs:

- Armaan Garg, 2019CSZ0002@iitrpr.ac.in
- Rahul Kumar Rai, 2018csz0004@iitrpr.ac.in

Your submission must be your original work. Do not indulge in any kind of plagiarism or copying. Abide by the honour and integrity code to do your assignment.

As mentioned in the class, late submissions will attract penalties.

Penalty Policy: There will be a penalty of 20% for every 24 hr delay in the submission. E.g. For the 1st 24 hr delay the penalty will be 20%, for submission with a delay of >24 hr and < 48 hr, the penalty will be 40% and so on.

You submission must include:

- A legible PDF document with all your answers to the assignment problems, stating the reasoning and output.
- A folder named 'code' containing the scripts for the assignment along with the other necessary files to run yourcode.
- A README file explaining how to execute your code.

Naming Convention:

Name the ZIP file submission as follows:
Name_rollnumber_Assignmentnumber.zip
E.g. if your name is ABC, roll number is 2017csx1234 and submission is for lab1 then you should name the zip file as: ABC_2017csx1234_lab1.zip

Assignment: Solving Sudoku

In this assignment, you will design an AI agent to solve the Sudoku puzzle. The core technique to focus on is the Constraint Satisfaction Problem (CSP). You will be implementing two methods to solve the Sudoku puzzle and compare the implementations. Some basics about the Sudoku puzzle and the CSP approach for solving the same are given below.

Introduction

Sudoku is a Japanese word for "single numbers", and refers to a numerical puzzle game that has become popular in newspapers and game publications all over the world. Although the rules for this puzzle are simple to understand, the solutions can range from the very simple to the agonizingly difficult.

Puzzle Background

The Sudoku puzzle has a 9x9 grid with some of the positions filled with digits between 1 to 9 to ensure a solution can be reached. The goal is to find and fill remaining cells with digits such that each row, column and the 3x3 square (sub-grid) all must contain the digits from 1 to 9, exactly once.

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

Solving Sudoku as Constraint Satisfaction Problems

A constraint satisfaction problem (CSP) consists of

- a set of variables,
- a domain for each variable, and
- a set of constraints.

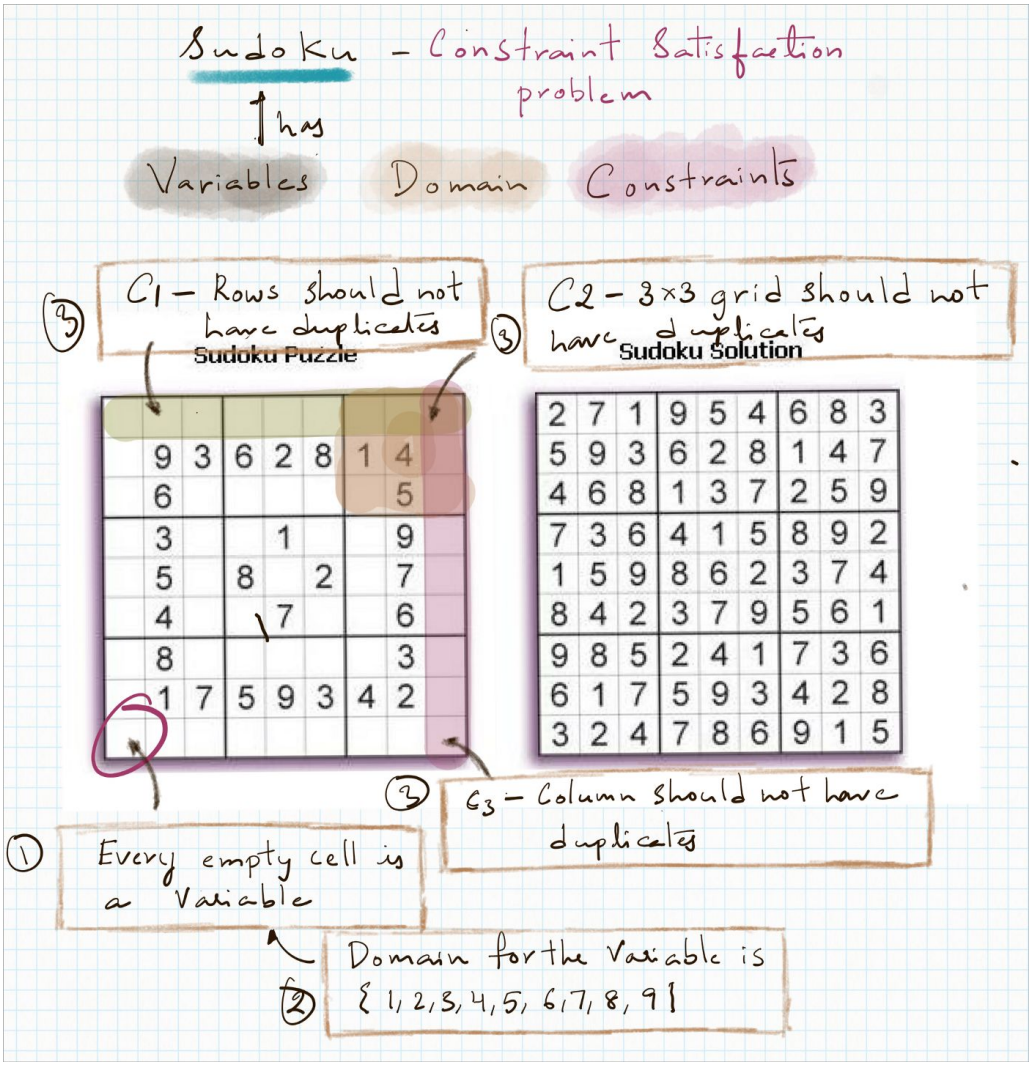
The aim is to choose a value for each variable such that the resulting possible world satisfies the constraints; we want a model of the constraints. The method in CSP should use the **constraint propagation** approaches as the inference to reduce the domain of each variable while using the **Backtracking search** with other suitable heuristics in order to find a solution.

Another approach that you have to consider is the Local Search based method - The MIN_CONFLIT algorithm. You can consider improving the Local Search method by incorporating Tabu Search.

Sudoku as CSP:

- Every empty cell is a variable
- Domain of each variable is {1,2,3,4,5,6,7,8,9}
- Constraints:
    - C1: Rows should not have duplicates
    - C2: Columns should not have duplicates
    - C3: 3x3 sub-grid should not have duplicates

Illustrations:



IMPLEMENTATION INSTRUCTIONS

Grid Layout

Your program will read the layout for each Sudoku puzzle from the input file that contains a 9x9 matrix of single characters. The characters will be the digits from 1 to 9 inclusive, plus the '0' character for any unassigned cell, where all of the cells are separated by space characters. For example, see below:

```
0 4 0 0 0 0 1 7 9
0 0 2 0 0 8 0 5 4
0 0 6 0 0 5 0 0 8
0 8 0 0 7 0 9 1 0
0 5 0 0 9 0 0 3 0
0 1 9 0 6 0 0 4 0
3 0 0 4 0 0 7 0 0
5 7 0 1 0 0 2 0 0
9 2 8 0 0 0 0 6 0
```

[PART 1] Backtracking Search with Constraint Propagation **[25 marks]**

[PART 2] Min_Conflict Search **[20 marks]**

FILES TO SUBMIT

(i) SudokuSolver_Backtracking.py

(ii) SudokuSolver_minconflict.py

the Python file containing the program that completes the objectives.

- 'input.txt' will be the name of the file containing the Sudoku puzzle.
- 'solution.txt' will be the name of the file wherein the solved Sudoku puzzle must be written.

NOTE: Calculate results for 5 different puzzles in the input.txt file. Code should be generic as it would be tested on various grid puzzles during evaluation.

Find the 5 test cases in the appendix at the bottom of the document.

[Other Python files may be submitted as well, if needed. Be sure to document your code thoroughly, as marks will be deducted for poor or unreadable designs. Graphical interface design is encouraged but not required.]

State the time and space complexity for each algorithm in your report. In your report, provide a table that compares performance of the above two algorithms for each 5 input cases..
Performance metrics: The total clock time, the search clock time, the number of nodes generated, N. Analyze your results to see if the behaviour you expected has been achieved or not, and why.
[5 marks]

Example on input matrix representation:

6				4			2	
				9				
					2		6	
					4	7		
			6					
1		2						
			7					
			5		6			4

(i) Sudoku puzzle grid test case

```
6 0 0 0 4 0 0 2 0
0 0 0 0 0 0 0 0 0
0 0 0 0 9 0 0 0 0
0 0 0 0 0 2 0 6 0
0 0 0 0 0 4 7 0 0
0 0 0 6 0 0 0 0 0
1 0 2 0 0 0 0 0 0
0 0 0 7 0 0 0 0 0
0 0 0 5 0 6 0 0 4
```

(ii) Representation in matrix form in input text file

Soution.txt file will be in the same 9x9 matrix form as input.txt with '0's replaced with the correct number.

Appendix

Test cases:

Test case 1:

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

Test case 2:

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

Test case 3:

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

Test case 4:

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

Test case 5:

6				4			2	
				9				
					2		6	
					4	7		
			6					
1		2						
			7					
			5		6			4

Good Luck!!!