CSC384 Intro to Artificial Intelligence

Assignment 2: Sudoku Solver

Introduction

Sudoku is the Japanese word for "single numbers", and refers to 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.

This assignment will require you to develop a Sudoku solver by applying search algorithms with heuristics to solve a puzzle.

Puzzle Background

The Sudoku puzzle is a 9x9 grid of squares, some of which contain number values from the start. Your goal is to add additional digits so that each row, column and 3x3 square contains the digits 1 to 9, inclusive.

Solving Sudoku

The brute force method of solving Sudoku involves a pencil and a large eraser. Filling in the squares one at a time with number values, you continue until one of the rows, columns or 3x3 squares contains a duplicate value.

	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	

Fig. 1 Sudoku example puzzle

This method can take a while, and is NP-complete for the generalized nxn puzzle case.

More elegant techniques exist of course, usually involving heuristics of some sort. A well-formed Sudoku puzzle is one that has a unique solution, so with careful analysis of the numbers in the grid, the solution can be determined without a single use of the eraser.

Objective

To create a Python program that takes in an incomplete Sudoku grid and returns the same grid with all the completed values. The assignment will be required to use some search algorithms to solve a puzzle, and return the puzzle solution, as follows:

- brute force (exhaustive search) method
- back-tracking (Constraint Satisfaction Problem (CSP)
- forward-checking with Mininum Remaining Values (MRV) heuristics

State the time/space complexity for each algorithm in your output result and report. In your report, provide a table that compares performance of the above three algorithms for each test cases listed in the appendix: 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. For algorithm bonus, you are encouraged to implement algorithms beyond the mininum required three algorithms above. You will receive 10 bonus points for extending/exploring another algorithm. You must describe and analyze your algorithm performance extended to the table required above. For some interesting heuristics, you can find some ideas online, and from the Russell & Norvig's book in Chapter 6.

Grid Layout

Your program will read the layout for each Sudoku puzzle from a 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, the file contents for the grid layout in Fig. 1 are show in the box (see Fig. 2).

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

Fig. 2 Input file 'puzzle.txt'

Instructions

You must develop your program and write your project report independently. (See course website policy for Plagiarism.)

The main class (containing the main function) **must be named as SudokuSolver.py**. Projects that do not compile will receive very little credit; and projects that work only on your machine will not receive full credit.

Submit SudokuSolver.py, the Python file containing the program that completes the objectives.

- 'puzzle.txt' is the name of the file in which the Sudoku layout can be found
- 'solution.txt' is the name of the file into which the completed Sudoku puzzle is written

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. You will be given 5 bonus points, if your Suduko can visually demonstrate the search process dynamically and clearly.

Without interface, your program **must** take command line arguments: a Sudoku puzzle file and the algorithm name (BF for brute force search, BT for back-tracking, FC-MRV for forward-checking with minimum remaining values.) The command line arguments must follow this order of: first, puzzle file name, then the algorithm name, for example:

```
Python SudokuSolver puzzle1.txt BF
Python SudokuSolver puzzle2.txt BT
Python SudokuSolver puzzle3.txt FC-MRV
```

Example of input puzzle given below:

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

Your program output to the screen must follow the format as follows: 1) display the solution, replacing 0s with the correct digits; 2) output the running time (milliseconds) and the number of nodes generated. You **must** also save these two specifications into two files respectively: the solution, for example for 'puzzle1.txt', as 'solution1.txt'; and the time and node evaluation as 'performance1.txt'. (Note that the values of the performance statistics given in the example are randomly generated and should not be taken as a reference.)

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

Total clock time: 1000.00

Search clock time: 800.00

Number of nodes generated: 500

Total clock time is measured from the program starts until the search process either finds a solution or fails. Search clock time records starting from when the search process starts to when the search process ends with a solution (or failure).

All programs will be run in the same environment and be compared to total run clock time on a random Sudoku puzzle. You will be rewarded with 8 bonus points as the fastest Sudoku solver; 5 bonus points for the second; and 3 bonus points for the third.

Submission

Submit your project through MarkUs:

https://mcsmark.utm.utoronto.ca/csc384_s15

You must submit two files: 'SudokuSolver.py', 'report.pdf'

In addition to the python code submission, a short report (maximum 2 pages, 12pt font, single-spaced), named as 'report.pdf', will also be required electronically. The report will describe different algorithoms, analyze the results and discuss the performance comparison. Also include in your report those design details and comments that are pertinent for the marker to know when reading your program.

Your program will be required to generate and save 'solution1.txt'~'solution5.txt', 'performance1.txt'~'performance5.txt' correspoding to the test case files 'puzzle1.txt'~'puzzle5.txt' in Appendix.

Assignment 2 is due on **March 6, Friday, at noon**.

Evaluation

Total points for this project is 100. The breakdown lists as follows:

- correctness of your solver to return a complete solution for a given puzzle (40%)
    - brute force (exhaustive search) method (5%)
    - back-tracking (CSP) (15%)
    - forward-checking with MRV heuristics (20%)
- extended heuristic algorithm (10 bonus points)
- performance analysis on the time/space complexity and discussion on the reason why algorithm improves (30%)
- Clarity, logical organization and comprehensive of technical report on alrogithm results & analysis (30%)

- Speedy Sudoku solver (8/5/3 bonus points)
- Graphical visulization interface design (5 bonus points)

Solution to example puzzle:

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

Appendix

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

(a) Sudoku puzzle test case 5

```
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
```

(b) input file puzzle5.txt representation

Hints and Tips

The website http://www.sudoku.com has many example grids to use in testing your design.