Assignment 3

Name: Diwakar Yalpi

UIN: 01127089

Github link: https://github.com/diwakaryalpi/AI-Assignment3

How to compile and execute

I have written two Python programs. One is basic sudoku solver that uses naive backtracking algorithm where the selection of variables and assignment of values can be done in order. The other sudoku solver that uses smart backtracking algorithm that is a combination of minimum remaining values (MRV), forward checking and used the naive backtracking while implementing the backtracking algorithm. Both the programs consider the input puzzle and parses the sudoku data file.

I have considered the sample inputs from the link: http://www.websodoku.com. I have considered easy, medium, hard and evil puzzles two each and saved all the puzzles in text file with the name sampleinputs.txt. The Python program considers the input from the sudokuinput.txt file. The puzzle must be updated in this file each time to try different puzzles.

```
python <python_filename>
```

In this case there are 2 filenames:

a. For Naive backtracking algorithm: filename will be naivebacktrackingsolver.py

b. For smart backtracking algorithm: filename will be smartbacktrackingsolver.py

one new line should be provided after the input puzzle so that the program understands that complete input is considered.

Naive Backtracking Algorithm

The Naive backtracking method is implemented by traversing through each empty space in the sudoku puzzle and checking for the valid values (for numbers between 1 and 9). Wrote a function to check the validity of values row-wise and column-wise, if the particular value cannot be placed because of the other constraints then the program backtracks to the previous position where another value was already checked for validity. This program traverses' row-by-row in search of empty spaces on sudoku puzzle. This takes a lot of branching hence is the name naive method.

Smart Backtracking Algorithm

For improving the Naive backtracking algorithm that's programmed previously, I have used minimum remaining values (MRV), Forward checking and used the same backtracking mechanism same as the naive backtracking algorithm. Below is the way that I have implemented MRV and Forward Checking:

Minimum Remaining values (MRV): By definition, it is choosing the variable with the fewest possible values. I have defined a function that select the next empty space based on the definition of the MRV. Filling the spot with a value from the domain results in an update of domains for all the blocks which are related to the former box. Remove the currently filled number from the domains of all the boxes in the same row, column and sub-box. This heuristic chooses the variable with the fewest remaining legal values to assign next.

Forward Checking: Forward checking detects the inconsistency earlier than simple backtracking and thus it allows branches of the search tree that will lead to failure to be pruned earlier than with simple backtracking. This reduces the search tree and (hopefully) the overall amount of work done. While solving sudoku puzzle, there can be possibility where there are no legal moves meaning that the space will be empty. So we are using this method so as to ensure that if we go through any branch from this position by storing the remaining legal moves of all blocks and updating them every time. There are intial checks to see if any variable has zero legal moves after the constraints are added.

The smart backtracking algorithm performed way better than the naive backtracking algorithm for most of the sudoku puzzles.

Results and Analysis:

The runtime of Naive backtracking algorithm is more than double in most of the cases when compared to the smart backtracking algorithm. The run times of Naive and smart backtracking algorithms is as follows:

Type of puzzle	Naive Backtracking Algorithm	Smart Backtracking Algorithm
Professor's puzzle	0.0041	0.0044
Easy puzzle	0.0063	0.0032
Medium puzzle	0.0088	0.0031
Hard puzzle	0.0521	0.0033
Evil puzzle	0.1571	0.0087

I have considered 2 puzzles but have taken only 1 puzzle in each category (Easy, Medium, Hard, Evil).

There are few observations when both Naive and smart backtracking algorithms are run on easy, medium, hard and evil puzzles. The number of backtracking steps for smart backtracking is less for most of the puzzles and also run time of the smart backtracking algorithm is less when compared to the Naive backtracking algorithm. Both Naive and smart backtracking algorithms work fine for both the programs. So, implementing minimum remaining values (MRV) and Forward checking heuristics has significantly improved the performance of Naive backtracking algorithm.