15 Puzzle Problem

Ryan Hiltabrand

Task 1

Formulate the Sudoku puzzle as a constraint satisfaction problem

- Variables
  - Every empty cell (cells containg 0)
- Domain
  - {1,2,3,4,5,6,7,8,9}
- Constraints
  - Rows should not contain duplicates
  - Columns should not contain duplicates
  - Each 3 by 3 Grid should not contain duplicates

About

...

Program Requirements

The program was written and tested with python 3.9 but should be able to run on any verision > 3.7. There are three different external libraries used: numpy, timeit, and tracemalloc. Install with the following commands:

- Python Version > 3.7
- python Packages
  - numpy (2d Arrays)
  - time (record program time)
  - copy (create deep copies of the domain so it does not get written over)

Instructions

1. Download and unzip folder
2. Navigate to the directory/folder with main.py
3. Install required packages `sh pip3 install [name of package]`
   - Get rid of brackets when you enter the name of package
4. Execute code through your operating systems terminal
   - Windows `sh python3 ./main.py`
   - Linux/Mac `sh python3 main.py`
5. Follow instructions on screen.
6. Find output in results directory

Program Execution

This program can be run in either 8 puzzle mode or 15 puzzle mode and relies on user input to run.

1. Asks user for difficulty and how it wants it to be solves
   - Difficulty
     - `easy`
     - `medium`
     - `hard`
     - `evil`
   - Type
     - `BT`
     - `SBT`
2. Program executes till solution is found or not using recursion

Analysis

This running this in the two seperate modes you notice that when including forward checking and minimum remaining value that execution time signifigantly decreases in all cases above easy. The biggest change occurs in the evil test as it took about 13 secounds for naive backtracking but with smart backtracking it becomes a mere 2.4 seconds. While with the Easy test the time maintained pretty close the amount of backtracks reduced from 98 to 0 due to forward checking. This decrease can be seen even more prevelant in the other cases as it went from 728->16, 869->10, and 16256->692. Specifically we can see that a large overhead comes by reducing the domains of each value ahead of time as well as choosing the one with the least amount in the domain.

Easy

```
easy = [[0, 3, 0, 0, 8, 0, 0, 0, 6],
        [5, 0, 0, 2, 9, 4, 7, 1, 0],
        [0, 0, 0, 3, 0, 0, 5, 0, 0],
        [0, 0, 5, 0, 1, 0, 8, 0, 4],
        [4, 2, 0, 8, 0, 5, 0, 3, 9],
        [1, 0, 8, 0, 3, 0, 6, 0, 0],
        [0, 0, 3, 0, 0, 7, 0, 0, 0],
        [0, 4, 1, 6, 5, 3, 0, 0, 2],
        [2, 0, 0, 0, 4, 0, 0, 6, 0]]
```

Results

```
 Backtracking:
[[7 3 4 5 8 1 2 9 6]
 [5 8 6 2 9 4 7 1 3]
 [9 1 2 3 7 6 5 4 8]
 [3 6 5 7 1 9 8 2 4]
 [4 2 7 8 6 5 1 3 9]
 [1 9 8 4 3 2 6 5 7]
 [6 5 3 9 2 7 4 8 1]
 [8 4 1 6 5 3 9 7 2]
 [2 7 9 1 4 8 3 6 5]]
Number of Backtracks:  98
--- 0.12560725212097168 seconds ---

Smart Backtracking:
[[7 3 4 5 8 1 2 9 6]
 [5 8 6 2 9 4 7 1 3]
 [9 1 2 3 7 6 5 4 8]
 [3 6 5 7 1 9 8 2 4]
 [4 2 7 8 6 5 1 3 9]
 [1 9 8 4 3 2 6 5 7]
 [6 5 3 9 2 7 4 8 1]
 [8 4 1 6 5 3 9 7 2]
 [2 7 9 1 4 8 3 6 5]]
Number of Backtracks:  0
--- 0.09508156776428223 seconds ---
```

Medium

```
medium = [[3, 0, 8, 2, 9, 6, 0, 0, 0],
          [0, 4, 0, 0, 0, 8, 0, 0, 0],
          [5, 0, 2, 1, 0, 0, 0, 8, 7],
          [0, 1, 3, 0, 0, 0, 0, 0, 0],
          [7, 8, 0, 0, 0, 0, 0, 3, 5],
          [0, 0, 0, 0, 0, 0, 4, 1, 0],
          [1, 2, 0, 0, 0, 7, 8, 0, 3],
          [0, 0, 0, 8, 0, 0, 0, 2, 0],
          [0, 0, 0, 5, 4, 2, 1, 0, 6]]
```

Results

```
 Backtracking:
[[3 7 8 2 9 6 5 4 1]
 [9 4 1 7 5 8 3 6 2]
 [5 6 2 1 3 4 9 8 7]
 [4 1 3 6 8 5 2 7 9]
 [7 8 9 4 2 1 6 3 5]
 [2 5 6 3 7 9 4 1 8]
 [1 2 4 9 6 7 8 5 3]
 [6 9 5 8 1 3 7 2 4]
 [8 3 7 5 4 2 1 9 6]]
Number of Backtracks:  728
--- 0.6275386810302734 seconds ---

Smart Backtracking:
[[3 7 8 2 9 6 5 4 1]
 [9 4 1 7 5 8 3 6 2]
 [5 6 2 1 3 4 9 8 7]
 [4 1 3 6 8 5 2 7 9]
 [7 8 9 4 2 1 6 3 5]
 [2 5 6 3 7 9 4 1 8]
 [1 2 4 9 6 7 8 5 3]
 [6 9 5 8 1 3 7 2 4]
 [8 3 7 5 4 2 1 9 6]]
Number of Backtracks:  16
--- 0.17114615440368652 seconds ---
```

Hard

```
 hard = [[7, 0, 0, 0, 0, 0, 0, 0, 0],
        [6, 0, 0, 4, 1, 0, 2, 5, 0],
        [0, 1, 3, 0, 9, 5, 0, 0, 0],
        [8, 6, 0, 0, 0, 0, 0, 0, 0],
        [3, 0, 1, 0, 0, 0, 4, 0, 5],
        [0, 0, 0, 0, 0, 0, 0, 8, 6],
        [0, 0, 0, 8, 4, 0, 5, 3, 0],
        [0, 4, 2, 0, 3, 6, 0, 0, 7],
        [0, 0, 0, 0, 0, 0, 0, 0, 9]]
```

Results

```
 Backtracking:
[[7 2 5 3 6 8 1 9 4]
 [6 8 9 4 1 7 2 5 3]
 [4 1 3 2 9 5 7 6 8]
 [8 6 7 9 5 4 3 2 1]
 [3 9 1 6 8 2 4 7 5]
 [2 5 4 1 7 3 9 8 6]
 [1 7 6 8 4 9 5 3 2]
 [9 4 2 5 3 6 8 1 7]
 [5 3 8 7 2 1 6 4 9]]
Number of Backtracks:  869
--- 0.7581503391265869 seconds ---

Smart Backtracking:
[[7 2 5 3 6 8 1 9 4]
 [6 8 9 4 1 7 2 5 3]
 [4 1 3 2 9 5 7 6 8]
 [8 6 7 9 5 4 3 2 1]
 [3 9 1 6 8 2 4 7 5]
 [2 5 4 1 7 3 9 8 6]
 [1 7 6 8 4 9 5 3 2]
 [9 4 2 5 3 6 8 1 7]
 [5 3 8 7 2 1 6 4 9]]
Number of Backtracks:  10
--- 0.17815256118774414 seconds ---
```

Evil

```
evil = [[0, 6, 0, 8, 0, 0, 0, 0, 0],
       [0, 0, 4, 0, 6, 0, 0, 0, 9],
       [1, 0, 0, 0, 4, 3, 0, 6, 0],
       [0, 5, 2, 0, 0, 0, 0, 0, 0],
       [0, 0, 8, 6, 0, 9, 3, 0, 0],
       [0, 0, 0, 0, 0, 0, 5, 7, 0],
       [0, 1, 0, 4, 8, 0, 0, 0, 5],
       [8, 0, 0, 0, 1, 0, 2, 0, 0],
       [0, 0, 0, 0, 0, 5, 0, 4, 0]]
```

Results

```
 Backtracking:
[[2 6 7 8 9 1 4 5 3]
 [5 3 4 2 6 7 8 1 9]
 [1 8 9 5 4 3 7 6 2]
 [3 5 2 1 7 4 9 8 6]
 [4 7 8 6 5 9 3 2 1]
 [6 9 1 3 2 8 5 7 4]
 [7 1 3 4 8 2 6 9 5]
 [8 4 5 9 1 6 2 3 7]
 [9 2 6 7 3 5 1 4 8]]
Number of Backtracks:  16256
--- 13.006664514541626 seconds ---

Smart Backtracking:
[[2 6 7 8 9 1 4 5 3]
 [5 3 4 2 6 7 8 1 9]
 [1 8 9 5 4 3 7 6 2]
 [3 5 2 1 7 4 9 8 6]
 [4 7 8 6 5 9 3 2 1]
 [6 9 1 3 2 8 5 7 4]
 [7 1 3 4 8 2 6 9 5]
 [8 4 5 9 1 6 2 3 7]
 [9 2 6 7 3 5 1 4 8]]
Number of Backtracks:  692
--- 2.3720366954803467 seconds ---
```