Artificial Intelligence Sudoku Solver

Apoorva Kaushik
April 20, 2022
CS 4613 Project 2

Running the solver

1. Download `sudoku_solver.py`
2. Place the `sudoku_solver.py` file in the same directory as formatted input files that will be solved.
3. Open your shell to the directory where the sudoku_solver.py file and input files are located.
4. Run the following command in the terminal:

   ```
   python3 sudoku_solver.py <input_file_name>
   ```

   Alternatively, you can run the following command in the terminal:

   ```
   python3 sudoku_solver.py
   ```

   After doing so you will need to type in the name of the file when prompted to do so.

   You can also run the code using your prefered IDE for Python.

5. Files containing the solved sudoku will be created in the same directory as the sudoku_solver.py file as follows:

   ```
   Output#.txt
   ```

Note: The input file should be named Input#.txt where # is the number of the input file. This will produce a correlated output file named Output#.txt.

Constraint Satisfaction Formulation

I set up sudoku with the following constraint satisfaction problem definition.

Variables: There are 81 variables, one for each cell in the sudoku. The variables are stored in a 2D array and refered to as `board[i][j]` where i is the row and j is the column of the cell.

Domains: Each variable in the sudoku board has a domain `1-9`.

Constraints: The following constraints are used to define the sudoku problem.

```
1. Alldiff(row)
2. Alldiff(column)
3. Alldiff(box)
```

Outputs

Output1.txt

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

Output2.txt

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

Output3.txt

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

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

Source Code

```
#!/usr/bin/env python3
"""
Apoorva Kaushik
April 20, 2022
CS 4613 Project 2
Sudoku Solver
"""

import argparse
import re
import copy

DOMAIN = [str(i) for i in range(1, 10)]

def string_board(sudoku_board):
    """
    parameters:
        sudoku_board (list): A nested list of a sudoku problem
    returns:
        str: A string representation of the sudoku board

    Prints the sudoku board in required format
    """
    return '\n'.join([' '.join(entry) for entry in sudoku_board])


def is_complete(state: list) -> bool:
    """
    Parameters:
        state (list): A nested list of a sudoku problem
    Returns:
        bool: True if the board is complete, False otherwise

    Checks if the sudoku board is completely filled in.
    Does NOT check legality since is_consistent is called for each cell
    """
    for row in state:
        if '0' in row:
            return False
    return True

def degree_heuristic(state: list, variables: list) -> tuple:
    """
    Parameters:
        sudoku_board (list): A nested list of a sudoku problem
```

```
        variables (list): A list of variables
    Returns:
        tuple: A tuple of the row and column of the empty cell

    Selects the variable with the most unassigned neighbors in row, col, and box
    """
    max_degree = -1
    max_var = None
    for var in variables:
        degree = 0
        for i in range(9):
            if state[var[0]][i] == '0':
                degree += 1
            if state[i][var[1]] == '0':
                degree += 1
        row_start = (var[0] // 3) * 3
        col_start = (var[1] // 3) * 3
        for i in range(row_start, row_start + 3):
            for j in range(col_start, col_start + 3):
                if state[i][j] == '0':
                    degree += 1
        if degree > max_degree:
            max_degree = degree
            max_var = var
    return max_var

def minimum_remaining_value(state: list) -> list:
    """
    Parameters:
        state (list): A nested list of a sudoku problem
    Returns:
        list: A tuple of the row and column of the empty cell(s) with fewest legal
values

    Finds the empty cell(s) in a sudoku board with fewest legal values possible
    """
    min_count = 10
    variables = []
    for i in range(9):
        for j in range(9):
            if state[i][j] == '0':
                count = 0
                for value in DOMAIN:
                    if is_consistent(state, i, j, value):
                        count += 1
                if count < min_count:
                    min_count = count
                    variables = [(i, j)]
                elif count == min_count:
                    variables.append((i, j))
    return variables

def select_unassigned_variable(state: list) -> tuple:
    """
```

```
    Parameters:
        state (list): A nested list of a sudoku problem
    Returns:
        tuple: A tuple of the row and column of the empty cell

    First runs the minimum remaining value heuristic and then degree heuristic to
break ties
    """
    minimum_variables = minimum_remaining_value(state)
    variable = degree_heuristic(state, minimum_variables)
    return variable

def is_consistent(state: list, row: int, col: int, num: int) -> bool:
    """
    Parameters:
        state (list): A nested list of a sudoku problem
        row (int): The row index
        col (int): The column index
        num (int): The number to be checked
    Returns:
        bool: True if the number is consistent, False otherwise

    Checks if the given number is consistent with the potential row, column, and
box
    """
    for i in range(9):
        if state[row][i] == num:
            return False
        if state[i][col] == num:
            return False
    row_start = (row // 3) * 3
    col_start = (col // 3) * 3
    for i in range(row_start, row_start + 3):
        for j in range(col_start, col_start + 3):
            if state[i][j] == num:
                return False
    return True

def backtrack(csp: list, state: list) -> list:
    """
    parameters:
        csp (list): A nested list of original sudoku problem
        state (list): A nested list of the sudoku's current state
    returns:
        list: A nested list of the solved sudoku problem

    Backtracking algorithm to solve sudoku and return the solved state
    """
    if is_complete(state):
        return state
    var = select_unassigned_variable(state)
    for value in DOMAIN:
        if is_consistent(state, var[0], var[1], value):
            state[var[0]][var[1]] = value
```

```
            if backtrack(csp, state):
                return state
            state[var[0]][var[1]] = '0'
    return None

def read_from_file(filename: str) -> list:
    """
    Parameters:
        filename (str): A file name as a string
    Returns:
        list[list]: A nested list of a sudoku problem

    Reads the input file and returns a nested list of the sudoku problem
    """
    board = []
    with open(filename) as file_pointer:
        for _ in range(9):
            board.append((file_pointer.readline().strip()).split())
    return board

def main():
    """
    This is the driver code for the sudoku solver. It will choose the file and
execute the solver on it.
    """

    parser = argparse.ArgumentParser(description='Sudoku Solver')
    parser.add_argument('in_file', nargs='?', help='Input file', default=None)
    cmd = parser.parse_args()
    filename = input("Enter File Name: ") if (not cmd.in_file) else cmd.in_file

    # regex to name the output file based on the input
    temp = re.findall(r'\d+', filename)
    res = list(map(int, temp))
    output_filename = "Output" + str(res[0]) + ".txt"

    # read from file and solve the sudoku before writing it to the output
    csp = read_from_file(filename)
    state = copy.deepcopy(csp)
    final = backtrack(csp, state)
    with open(output_filename, "w") as out_file:
        out_file.write(string_board(final))

if __name__ == "__main__":
    main()
```