Sudoku Solver using AI

Report 2

submitted by

Lakshmipriya Ragupathi

CS21B2001
B. Tech. Computer Science and Engineering with major in AI
Department of Computer Science

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY,
DESIGN AND MANUFACTURING, KANCHEEPURAM
CHENNAI-600127
APRIL 2023

TABLE OF CONTENTS

Sudoku using Exhaustive Search

The Sudoku puzzle is typically represented by a 9x9 grid, consisting of 81 squares. Each square is either filled with a number from 1-9 or left blank, which is represented by the number 0. The rules of the puzzle:

1. Each sub-grid of size 3x3 contains the numbers from 1-9 wherein each digit appears only once.
2. Each digit appears only once in its row and column.

`

The solution can be reached using the following steps or procedures:

1. Find the next empty cell to fill : This can be done by linearly searching for “0” in the matrix
2. Fill the empty space with a number from the array [1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9]
3. Check the validity of the board: This can be done by checking the row, column and 3x3 sub-grid and check if the current cell contains unique number or not. If the number is not unique, go back to Step 2 (recurring) and choose a different number.
4. Repeat the steps.

After filling a cell with a number, the player must check the validity of the board. This involves verifying that the current cell's value is unique in its corresponding row, column, and sub-grid. If the number is not unique, the player must go back to Step 2 and choose a different number. The backtracking can be seen in the following example:

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

1. We start at the first empty cell which is at (0,2) and fill it with 1.
2. Check the validity of the board, it passes.
3. Move to the next empty cell at (0,3) and fill it with 2.
4. Check the validity of the board, it passes.
5. Move to the next empty cell at (0,5) and fill it with 3.
6. Check the validity of the board, it fails because 3 already appears in the same column.
7. Change the number at (0,5) to 4 and check the validity of the board again. It passes.
8. Move to the next empty cell at (0,6) and fill it with 3.
9. Check the validity of the board, it passes.
10. Move to the next empty cell at (0,7) and fill it with 5.
11. Check the validity of the board, it fails because 5 already appears in the same row.
12. Change the number at (0,7) to 6 and check the validity of the board again. It fails.
13. Since the validity of the board has failed and 6 has to appear in the row, we need to backtrack to the previous step (Step 7).
14. Continue this process until the entire Sudoku grid is filled.

As you can see in the above example, we had to undo 7 moves and restart therefore, this algorithm is inefficient. However, it ensures that the Sudoku puzzle is solved correctly.

Although the backtracking algorithm is a reliable method for solving Sudoku puzzles, it has some limitations, particularly in terms of time complexity. The algorithm's time complexity is $O(9^m)$, where m is the number of empty cells in the grid. This means that as the number of empty cells in the grid increases, the algorithm's efficiency decreases significantly.

As a result, AI search algorithms have become increasingly popular in solving Sudoku puzzles. AI search algorithms are designed to make intelligent decisions about which numbers to place in each cell, based on a set of rules and heuristics. These algorithms can analyze the puzzle's complexity and determine the most efficient way to solve it.

One example of an AI search algorithm used to solve Sudoku puzzles is Simulated Annealing.

Sudoku using Simulated Annealing

Simulated annealing can be used to solve Sudoku puzzles, which involve filling in a 9x9 grid with numbers from 1 to 9 such that each row, column, and 3x3 sub-grid contains all the numbers from 1 to 9 exactly once.

To apply simulated annealing to Sudoku, the puzzle can be represented as a 9x9 grid of decision variables, where each variable represents a cell in the grid and takes a value from 1 to 9. The objective function is to minimize the number of violations of the Sudoku constraints, which occur when a row, column, or sub-grid contains duplicate values.

The simulated annealing algorithm starts with a random initial state, where each cell is assigned a random value from 1 to 9. The objective function is then evaluated to calculate the number of constraint violations.

The algorithm then iteratively perturbs the state by randomly selecting a cell and assigning it a new value. If the new state results in a lower number of constraint violations, the new state is accepted as the current state. However, if the new state results in a higher number of constraint violations, the new state is accepted with a probability that depends on the temperature and the size of the increase in constraint violations.

The temperature is initialized to a high value and gradually decreased over the course of the algorithm. This allows the algorithm to explore the search space broadly at the beginning, accepting many worse states to avoid getting stuck in local minima. As the temperature decreases, the algorithm becomes more selective and converges to a minimum.

The perturbation function used in simulated annealing for Sudoku can involve different strategies, such as randomly selecting a cell and trying different values until a valid assignment is found.

The algorithm continues iterating and perturbing the state until a stopping criterion is met, such as reaching a minimum number of constraint violations or running out of time. The final state represents a solution to the Sudoku puzzle, where each cell is assigned a value from 1 to 9 that satisfies all the Sudoku constraints.

Simulated annealing has some advantages for solving Sudoku puzzles. It can efficiently explore the search space and avoid getting stuck in local minima, which can be a problem for other optimization algorithms. Additionally, it can handle objective functions that are non-differentiable or have unknown structure, which is the case for the Sudoku constraints.

However, the performance of simulated annealing for Sudoku can depend on the perturbation function and the cooling schedule. If the perturbation function is not effective at exploring the search space or maintaining the Sudoku constraints, the algorithm may get stuck or produce invalid solutions. Similarly, if the cooling schedule is not properly tuned, the algorithm may converge too quickly or not find a good solution at all.

Exhaustive Search:

```
def solve (s0):
    i, j = Find_next_empty_cell(s0)
      if i == -1
            return True
    for k in [0,9]:
      try different values in (i,j)
      Check_Validity(s0, i, j, k)
      if valid:
            update
            if solve(s0, i, j):
                  return True
            otherwise undo


    return False
```

Other than the above functions, there will be other helper functions:

1. `validity (s):`
2. `Find_next_empty_cell (s):`
3. `print_board (s):`

Simulated Annealing:

```
def simulated_annealing(s0, k_max):
    s = s0
    for k in [0,k_max]:
        T = temperature(k/k_max)
        s_new = neighbour(s)
        if Probability(E(s), E(s_new), T) >= random.random():
            s = s_new
    return s
```

```
def neighbour (s):
    randomly change a cell's value to another value
    return s
```

Other than the above functions, there will be other helper functions:

4. `validity (s):`
5. `print_board (s):`

REFERENCES

1. “Enumerating possible Sudoku grids” *math.sc.edu,* https://ocw.mit.edu/courses/6-s095-programming-for-the-puzzled-january-iap-2018/resources/mit6_s095iap18_puzzle_8/

2. “Programming for the Puzzled: You Won’t Want to Play Sudoku Again” *ocw.mit.edu,* www.rhydlewis.eu/papers/META_CAN_SOLVE_SUDOKU.pdf