SUDOKU SOLVERS

GANDIKOTA SAI KRITHIKA

PES1UG20CS148
DEPT. OF COMPUTER SCIENCE
PES UNIVERSITY

BANGALORE,INDIA
krithikagandikota@gmail.com

Abstract*— *In the last decade, solving the Sudoku puzzle has become every one's passion. The simplicity of puzzle's structure and the low requirement of mathematical skills caused people to have enormous interest in accepting challenges to solve the puzzle. Therefore, developers have tried to find algorithms in order to generate the variety of puzzles for human players so that they could be even solved by computer programming. In this essay, I have presented an algorithm called the Backtracking algorithm using human strategies. The purpose is to implement a more efficient algorithm and then compare it with other general Sudoku solvers. This algorithm is a general algorithm that can be employed in to any problems. The results have proved that the Backtracking algorithm solves the puzzle faster and more effective than the brute force algorithm which is a very generic one.

Keywords— Backtracking algorithm, Sudoku, Brute force Algorithm, human strategy

Problem Statement/Motivation:

The Sudoku puzzle problem has been shown to be NP-complete, which severely limits the ability to solve sudoku puzzles with increasing complexity. For Sudoku puzzles with low order (3, 4) this is usually not an issue today, as even small handheld devices have enough computing power to solve the most difficult Sudoku puzzles in reasonable time. However, because of its NP-complete properties, finding efficient implementations of solvers may prove as proof-of-concept for other similar algorithms for other NP-complete problems. In this regard, finding efficient algorithms to solve Sudoku puzzles may set precedents for other algorithms.

This report presents a Sudoku solver known as the Backtracking method that uses straightforward rules to complete the problems. The algorithm is based on human methods and guessing. This indicates that the algorithm is applied depending on how people perceive things.

The effectiveness of the suggested algorithm is then assessed by comparison with the Brute Force algorithm.

I. SUDOKU PUZZLE

The structure of the puzzle is very simple, especially the classic puzzle. This essay is mainly focused on classic puzzle of a 9X9 grid. There already exist a number of digits in the board that make the puzzle solvable. It means that some numbers are already placed in the Sudoku board before starting. The board consists of 81 cells, which is divided into nine 3X3 sub boards and each 3X3 sub board is called "box" or "region". The main concept of the game is to place numbers from 1 to 9 on a 9X9 board so that every row, column and box contains all the numbers(1-9) but once, that is, the numbers are not repeated more than once.

II. BACKTRACKING ALGORITHM

The objective of backtracking, an algorithmic methodology, is to find every possible solution to a problem by applying the "brute force" method. It entails gradually compiling a set of each solution. A issue would have limits, so any solutions that don't meet them would be eliminated.

It uses recursive calling to find a solution set by building a solution step by step, increasing levels with time. In order to find these solutions, a search tree named state-space tree is used. In a state-space tree, each branch is a variable, and each level represents a solution.

The depth-first search strategy is employed by a backtracking algorithm. A bounding function is used when the algorithm begins exploring the solutions so that it may verify that the already developed solution satisfies the constraints. If it does, it keeps looking. If not, the algorithm would revert to the previous level and the branch would be removed.

III. SUDOKU USING BACKTRACKING

***PSEUDOCODE**:*

```
solve(grid)
if(no more cells are there to explore)
   return true;
for(all available choices)
   try one choice c;
   if(solve(grid with choice c made) is true)
         return true;
   unmake choice c;
return false;
```

sudokuSolver(grid)

1.Find an unfilled cell (i,j) in grid

2.If all the cells are filled then

2.1. A valid sudoku is obtained hence return true

3.For each num in 1 to 9

3.1. If the cell (i,j) can be filled with num then fill it with num temporarily to check

3.2. If sudokuSolver(grid) is true then return true

3.3. If the cell (i,j) can't be filled with num the mark it as unfilled to trigger backtracking

4.If none of the numbers from 1 to 9 can be filled in cell (i,j) then return false as there is no solution for this sudoku.

Explanation of the Algorithm:

-We start by finding an unfilled cell(i,j).

-If all the cells are filled then a valid sudoku is obtained.

-We try to place every number between 1 to 9 in the unfilled cell.

-Before placing we check for the constraints by checking the current row,current column and current 3×3 submatrix.

-If any of the any of the constraint becomes false we'll not place that number at (i,j).

-If all the constraints are true then we'll place that number at (i,j) and then repeat the process by finding an unfilled cell.

-If at any point none of the numbers can be placed at (i,j) then we've to backtrack and change the values for already visited cells

IV. BACKTRACKING ALGORITHM-TIME AND SPACE COMPLEXITY

Time complexity: O(9^(N*N)), For every unassigned index, there are 9 possible options so the time complexity is O(9^(n*n)). The time complexity remains the same but there will be some early pruning so the time taken will be much less than the naive algorithm but the upper bound time complexity remains the same.

O(n ^ m) where n is the number of possibilities for each square (i.e., 9 in classic Sudoku) and m is the number of spaces that are blank.

The problem can be designed for a grid size of N*N where N is a perfect square. For such an N, let M = N*N, the recurrence equation can be written as

T(M) = 9*T(M-1) + O(1)

where T(N) is the running time of the solution for a problem size of N. Solving this recurrence will yield, O(9^M).

Space Complexity: O(N*N), To store the output array a matrix is needed.

V. COMPARISON WITH OTHER ALGORITHMS

In this section I present the result of the testing and also examine the differences between the backtracking algorithm and other algorithms used in solving Sudoku such as brute force algorithm and heuristic search.

VI. BACKTRACKING vs BRUTE FORCE

A brute force algorithm visits the empty cells in some order, filling in digits sequentially, or backtracking when the number is found to be not valid, Briefly, a program would solve a puzzle by placing the digit "1" in the first cell and checking if it is allowed to be there. If there are no violations (checking row, column, and box constraints) then the algorithm advances to the next cell and places a "1" in that cell. When checking for violations, if it is discovered that the "1" is not allowed, the value is advanced to "2". If a cell is discovered where none of the 9 digits is allowed, then the algorithm leaves that cell blank and moves back to the previous cell. The value in that cell is then incremented by one. This is repeated until the allowed value in the last (81st) cell is discovered.

if we look the game row by row, for each row, we have N possible values for the first block, and $N-1$ possible values for the second block, $N-2$ possible values for the third block, and 1 value for the last block. Hence the computational complexity for filling one row is $O(N \times (N-1) \times \ldots \times 1) = O(N!)$. As there are N rows, the total computational complexity is $O((N!)^2)$. In fact, this can be further reduced, but it is hard to find a short formula to represent it, so we will just keep this as the final computational complexity of brute force search.
Thus, backtracking would be a better option than a brute force approach.

VII. BACKTRACKING vs HEURISTIC SEARCH

An alternative way to solve a Sudoku game is heuristic search. In every step, we try to find a block that has least possible values. The detailed steps are given as follows:

1. Scan all empty blocks, for every empty blocks, maintain a list of possible values by checking the row, column and inner boxes that contain the block.

2. Find a block that has least possible values.

3. If the block has only one possible value, assign that value to the block, and repeat from step 1. If the block has multiple possible values, assign one value from the list, and keep a record of which value has been assigned and all the other remaining values, then repeat from step 1. If the block has no possible value, go to step 4.

4. If there is one block that has multiple possible values, assign another value in the list and remove all the values after the block. If all values have been tested, check the second last block that has multiple possible values, until a solution is found, or no solution if all blocks having multiple possible values are checked. It is easy to see the steps of heuristic search share some commons with backtracking. The key difference is that heuristic search tries to minimize the cost of testing possible values by introducing more tests on all empty blocks. The introduced cost may be ignored because it is marginal. Therefore, if every empty block has exactly 0 or 1 possible value, it is the best scenario with computational complexity $O(N^2)$ as every block needs to be checked once only. But the worst scenario where all blocks are empty will have computational complexity the same as brute force search, which is $O((N!)^2)$.

Thus, the heuristic search seems to be way more optimised than the backtracking approach.

VIII.CONCLUSION AND FUTURE WORK:

Backtracking remains a valid and vital tool for solving various kinds of problems, even though this algorithm's time complexity may be high, as it may need to explore all existing solutions of time. It can further be improved/optimised by:
1.Trying different algorithms than the Backtracking such as Stochastic search, Constraint programming, Exact cover, and others.
2.Exploring the use of Machine Learning algorithms to solve Sudoku Puzzles using a dataset available on Kaggle which provides 9 million Sudoku Puzzle-Solution pairs.

3.Using Convolutional Neural Networks and Image processing to detect Sudoku Puzzles from the hard copies and then displaying the found solutions onto it using the AR (Augmented Reality) Technology.

IX. REFERENCES

[1] http://www.iraj.in/journal/journal_file/journal_pdf/12-625-15839037484-7.pdf

[2] https://github.com/tejasmorkar/sudoku-solver#future-works

[3] https://medium.com/dsckiit/build-a-sudoku-solver-using-backtracking-1c761bfcde58

[4] https://levelup.gitconnected.com/csp-algorithm-vs-backtracking-sudoku-304a242f96d0