

**SUDOKU SOLVER**

**DESIGN AND ANALYSIS OF ALGORITHMS**

(UE17CS251)

**BY:**

MEGHA M PES1201700948

KONA SUPRIYA PES1201701117

**SECTION H**

**ABSTRACT:**

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. The purpose is to implement a more efficient algorithm.

**ABOUT THE ALGORITHM:**

Usually, the brute force algorithm can be applied to any possible algorithm. In this case the algorithm goes through every empty square and places a valid digit in that square. If no valid number is found the algorithm comes back to the previous square and change the value in that square. The process is repeated until the board is filled with numbers from 1 to n.

The advantage of the brute force algorithm is that the algorithm can guarantee a solution to any puzzles since it generates all possible answers until the right answer is found if the puzzles are valid. Additionally, the running time can be unrelated to level of difficulty, because the algorithm searches for every possible solution.

The backtracking method, which is similar to the human strategy (guessing), is used as a help method to the pencil-and-paper algorithm. Generally, the backtracking method find empty square and assign the lowest valid number in the square once the content of other squares in the same row, column and box are considered. However, if none of the numbers from 1 to n are valid in a certain square, the algorithm backtracks to the previous square, which was filled recently.

Time complexity: O(N^(NxN))

Where N is the size of the matrix.

**HIGH-LEVEL ALGORITHM:**

RecursiveBacktracking(Puzzle[][])

{

Puzzle [][] //global

SolvePuzzle (row, col)//function

{

if (no more choices):

the puzzle is solved.

if (Puzzle [row][col]= notEmpty):

move to the next square.

for 1 to n:

if(CheckRow(row,col,digit) & CheckCol(row,col,digit) & CheckBox(row,col,digit)

{

Puzzle [row][col]= digit;

move to the next square

}

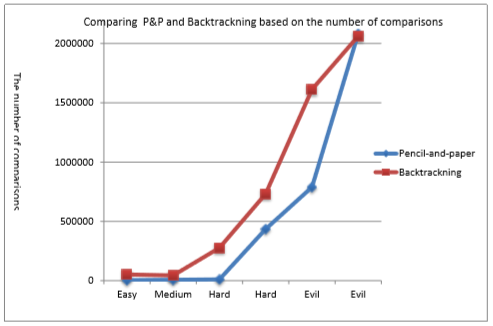
if not valid number is found go the previous square that was recently filled.

}

}

**TEST RESULTS:**

The brute force algorithm seems to be a useful method to solve any Sudoku puzzles and it can guarantee to find at least one solution.



**RUNNING RESULTS:**

**INPUT:**

Puzzle [9][9]={{0, 0, 0, 0, 0, 0, 0, 9, 0},

{1, 9, 0, 4, 7, 0, 6, 0, 8},

{0, 5, 2, 8, 1, 9, 4, 0, 7},

{2, 0, 0, 0, 4, 8, 0, 0, 0},

{0, 0, 9, 0, 0, 0, 5, 0, 0},

{0, 0, 0, 7, 5, 0, 0, 0, 9},

{9, 0, 7, 3, 6, 4, 1, 8, 0},

{5, 0, 6, 0, 8, 1, 0, 7, 4},

{0, 8, 0, 0, 0, 0, 0, 0, 0}}

OUTPUT:

-------------------

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

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

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

-------------------

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

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

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

-------------------

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

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

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

-------------------

0.001499 sec