REPORT FOR SUDOKU USING BACKTRACKING

As a project for course

ARTIFICIAL INTELLIGENCE (INT 404)

Name	**Irlapati Paavan Kumar**
Registration number	**11803029**
Semester	**Fourth**
School	**School of Computer Science and Engineering**
Name of the University	**Lovely Professional University**
Date of Submission	**31st March 2020**

Submitted To

SAGAR APNDEY(Asst. professor)
Lovely Professional University

Jalandhar, Punjab, India.

Solving Sudoku

31st March 2020

ACKNOWLEDGEMENT:-

I would like to thank my mentor - Prof. Sagar Pande for his advice and inputs on this project. Many thanks to my friends and seniors as well, who spent countless hours to listen and provide feedbacks.

Table of Contents

1.ABSTRACT:-

Solving sudoku focuses on how a generalized 9*9 sudoku can be solved using backtracking algorithm. This makes the sudoku to be solved in the shortest way possible as it avoids the traditional naïve method of solving with every possible Solution as it would end up giving some relatively 9*9*9 methods.

2.INTRODUCTION

2.1 CONTEXT

This project has been done as a part of my course for CSE at Lovely Professional University. Supervised by Sagar pande, I have three months to fulfill the requirements in order to succeed the module.

2.2 MOTIVATIONS

Being extremely interested in everything having a relation with Artificial Intelligence, the group project was a great occasion to give us the time to ,learn and confirm our interest for this field. The fact that we can make estimations, prediction and give the ability for machines to learn by themselves is both powerful and limitless in term of application possibilities. We can use Artificial Intelligence in Finance, Medicine, almost everywhere. That's why I decided to conduct my project around the Artificial Intelligence.

2.3 IDEA

As a first experience, we wanted to make my project as much simple as possible by approaching every different algortims used in Artificial Intelligence. The problems that are not immediate scientific interest but useful to illustrate and practice, we chose to take sudoku to solve using "Backtrackig" as approach.

The goal was to predict the empty columns of a given sudoku puzzle chart according to the already given elements in the puzzle taking into account different “conditions” that will be developed in the following .

3. MEMBER CONTRIBUTION

CONTRIBUTIONS

1. PAAVAN KUMAR IRLAPATI

1. Coding
2. Algorithmic part
3. Report

4. ALGORITHMS

BACKTRACKING ALGORITHM:-

For solving the current problem in this project that is, sudoku I chose backtracking algorithm which mainly works on the recursion technique.

This algoritmic technique is used for ssolving problems recursively by trying to build a sloution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point of time

Backtracking can ve defined as a general algorithmic techniwue that considers searching every possible combination in order to solve a computational problem

There are three types of problems in backtracking –

1.Decision problem :- we search for a feasible solution.
2. Optimization problem :- we search for the best solution.
3. Enumeration problem :- we find all feasible solutions.

5. SCOPE OF THE PROJECT :-

5.1 How it works?

The sudoku which is used for this particular program is an 9*9 sudoku consisting of 9 perfect 3*3 blocks.

As this programme focuses on solving the sudoku using backtracking algorithm; this program solving can be divide into three subdivisions where each division focuses on certain aspect of the problem.

1.our choice

2.the constraints.

3.the final goal

First the rules of sudoku are:

Numbers 1 through 9.

Each number can only work once in a row and column vice versa

And there are 3*3 sub-matrices.

We can traverse and place an entry in empty cells one by one either from columns or rows satisfying the rules of solving sudoku

We can check the whole board to see if the board is still valid

If the entry is a valid move, we can go through these repetitive steps

If the entry is found to be invalid, we don't follow that path anymore.

We come back to one step back and check for the possibilities if there can be any other chance.

When all the entries have been filled, we are done with the sudoku problem solving.

The three key aspects of solving these problems:

1.our choice

The entry which we place in an empty cell that we look

2. the constraints are:

Standard sudoku constraints. We cannot make a placement that will break the board.

3.our final goal

Place all entries on the empty spaces on the board.

We will know how we have done this when we have placed a valid value in the last cell; in our search which programmatically will be the bottom right cell in the 2-dimensional matrix.

The final most thing to do here is that validating all the placements are at the right place.

1. Find some empty space
2. Attempt to place the digits 1-9 in that space

3. Check if that digit is valid in the current spot based on the current board
4. a. If the digit is valid, recursively attempt to fill the board using steps 1-3.
   b. If it is not valid, reset the square you just filled and go back to the previous step.
5. Once the board is full by the definition of this algorithm, we have found a solution.

COMPLEXITY :- As every problem has complexities

Since sudoku is generally 9*9 we can't really discuss complexities because nothing can scale.

Solving sudoku generalized to n*n boards is NP-complete problem so we know for sure that our time complexity is exponential at the very least.

The scope of this project is to solve any generalized 9*9 sudoku with given entries in the columns.

For doing this one has to give the inputs once the program has been executed on asking. And the user has to give 81 entries each for each column for the whole matrix and '0' in the place of empty columns so as to satisfy the giving condition form the code

```
29  def search(chart):          #searches for the empty columns throughout the chart
30      for i in range(len(chart)):          #gives the position at which there are empty columns
31          for j in range(len(chart[0])):
32              if chart[i][j] == 0:
33                  return (i, j)
34          print(search(sudoku))
35      return None
```

As mentioned in the line number 32 it take the column as the 0 for empty column.

Line 34 prints the empty positions.

And the later function named check validates if an allocate number does satisfy for all the remaining numbers allocated in the other columns of the chart

First it checks for the rows and later columns followed by the 3*3 sub – matrices.

And the main function here which contains the algorithmic code which calls the above mentioned two functions hence doing bith the tasks of searching for an emoty space and checking for the aloocated number in the empthy column for each and every possible and when not satisfying the constraints going back to the initial state at which it gets another satisfactory condition.

```
def solve(chart):
    find = search(chart)        #calls search function
    if not find:
        return True
    else:
        horizontal, vertical = find

    for i in range(1,10):
        if check(chart, i, (horizontal, vertical)):
            chart[horizontal][vertical] = i
            if solve(chart):
                return True

            chart[horizontal][vertical] = 0

    return False
```

And this condition repeats i.e recursively till it solve or reaches the final goal or state.

6. SCREENSHOTS :-

Editor - C:\Users\paava\New folder\sudoku.py

sudoku.py

```
#taking the input from the user for the sudoku chart
sudoku = []
numberOfRows = 9
numberOfColumns = 9
print("enter each value and press enter ,and give 0 as value for empty characters")
for row in range (numberOfRows):
    sudoku.append([])            #adds an empty new row
    for column in range (numberOfColumns):
        value = eval(input())
        sudoku[row].append(value)


def display(chart):             #displays the sudoku chart based on the given input
    for i in range(len(chart)):
        if i % 3 == 0 and i != 0:       #after the third line and simultaneously for initial value
            print("- - - - - - - - - - - - ")     #not equal to 0 prints a dotted line

        for j in range(len(chart[0])):      #same as for the third column for vertical lines
            if j % 3 == 0 and j != 0:
                print(" | ", end="")

            if j == 8:                  #prints 8th index position 1-9 for every row
                print(chart[i][j])
            else:
                print(str(chart[i][j]) + " ", end="")
            #prints the remaining rows and columns
display(sudoku)

def search(chart):         #searches for the empty columns throughout the chart
    for i in range(len(chart)):         #gives the position at which there are empty columns
        for j in range(len(chart[0])):
            if chart[i][j] == 0:
                return (i, j)
        print(search(sudoku))
    return None


```

Editor - C:\Users\paava\New folder\sudoku.py

sudoku.py*

```
37
38
39 def check(chart, number, position):#checks if the allocated value does satisfy with the conditions are not
40     # Checking rows for the conditions
41     for i in range(len(chart[0])):#starting from the intitial index row from the list
42         if chart[position[0]][i] == number and position[1] != i:#allocating the number to the position
43             return False
44
45     # Check columns for the conditions
46     for i in range(len(chart)):
47         if chart[i][position[1]] == number and position[0] != i:
48             return False
49
50     # Checking the sub 3*3 matrix
51     subMatrix_row = position[1] // 3
52     subMatrix_column = position[0] // 3
53
54     for i in range(subMatrix_column*3, subMatrix_column*3 + 3):
55         for j in range(subMatrix_row * 3, subMatrix_row*3 + 3):
56             if chart[i][j] == number and (i,j) != position:
57                 return False
58
59     return True
60
61
```

```
61
62 #backtracking algorithm using recursive function
63 #callinfg for each and every element inserted at evry position by calling check function
64 def solve(chart):
65     find = search(chart)        #calls search function
66     if not find:
67         return True
68     else:
69         horizontal, vertical = find
70
71     for i in range(1,10):
72         if check(chart, i, (horizontal, vertical)):      #calls check function
73             chart[horizontal][vertical] = i
74             if solve(chart):                             #calls solve function
75                 return True
76
77             chart[horizontal][vertical] = 0
78
79     return False
80
81 solve(sudoku)
82 print("_____")
83 display(sudoku)
```

Permissions: RW End-

7. WORKING :-

This takes user input into an list comprising of 9 rows and 9 columns taking input for each and every posiible column

```
#taking the input from the user for the sudoku chart
sudoku = []
numberOfRows = 9
numberOfColumns = 9
print("enter each value and press enter ,and give 0 as value
for row in range (numberOfRows):
    sudoku.append([])            #adds an empty new row
    for column in range (numberOfColumns):
        value = eval(input())
        sudoku[row].append(value)
```

This display class works on converting the user input lists into an sudoku 9*9 chart by sudividing them into 3*3 matrix too.

```
def display(chart):              #displays the sudoku c
    for i in range(len(chart)):
        if i % 3 == 0 and i != 0:        #after the thi
            print("- - - - - - - - - - - - ")     #n

        for j in range(len(chart[0])):       #same as f
            if j % 3 == 0 and j != 0:
                print(" | ", end="")

            if j == 8:                   #prints 8th in
                print(chart[i][j])
            else:
                print(str(chart[i][j]) + " ", end="")
            #prints the remaining rows and columns
display(sudoku)
```

This check function with parameters chart, number, position

Helps in checking the validity of an allocated element in the chart that is board which is being referred as chart in this program

```
def check(chart, number, position):#checks if the allocated value
    # Checking rows for the conditions
    for i in range(len(chart[0])):#starting from the intitial inde
        if chart[position[0]][i] == number and position[1] != i:#a
            return False

    # Check columns for the conditions
    for i in range(len(chart)):
        if chart[i][position[1]] == number and position[0] != i:
            return False

    # Checking the sub 3*3 matrix
    subMatrix_row = position[1] // 3
    subMatrix_column = position[0] // 3

    for i in range(subMatrix_column*3, subMatrix_column*3 + 3):
        for j in range(subMatrix_row * 3, subMatrix_row*3 + 3):
            if chart[i][j] == number and (i,j) != position:
                return False

    return True
```

And the solve function calls both the check and search function recursively for each and every element being kept there in the empty columns for solving the puzzle.

The given puzzel on the top of the horizontal line and the solved answer below the horizontal line.

```
0 0 1   | 9 8 4   | 7 6 0
6 0 0   | 0 5 7   | 0 0 0
8 0 7   | 0 1 0   | 0 0 0
- - - - - - - - - - - - -
9 6 0   | 3 0 8   | 1 0 5
1 8 5   | 0 2 0   | 0 7 3
3 0 0   | 0 0 0   | 2 0 8
- - - - - - - - - - - - -
2 1 0   | 0 0 0   | 0 3 6
0 0 0   | 1 0 0   | 0 0 4
0 9 6   | 0 0 2   | 5 1 0
-------------------------
5 3 1   | 9 8 4   | 7 6 2
6 4 9   | 2 5 7   | 3 8 1
8 2 7   | 6 1 3   | 4 5 9
- - - - - - - - - - - - -
9 6 2   | 3 7 8   | 1 4 5
1 8 5   | 4 2 9   | 6 7 3
3 7 4   | 5 6 1   | 2 9 8
- - - - - - - - - - - - -
2 1 8   | 7 4 5   | 9 3 6
7 5 3   | 1 9 6   | 8 2 4
4 9 6   | 8 3 2   | 5 1 7
```

It solves optimal problems till an hard range sudoku puzzles .
(Note:- it can not solve expert categorized problems.)

Adding the code in line 64 prints each and every step how solving the problem goes on .

```
63 def solve(chart):
64     print(sudoku)
```

Refer the “step by step.txt” document on how solving the problem carries on for every step in the case of this current problem which is attached along with this report.

8. CONCLUSIONS :-

Solving sudoku can be done with the other algorithms and methods also for example naïve method which would take a lot of time and space for solving etc…, but considering backtracking algorithm gives the best possible solutionsaving time and space .
But for solving 9*9 sudoku puzzles here comes with some setbacks which are like consider this problem which is categorized as an expert puzzle

```
0 0 0  | 0 0 0  | 0 0 9
0 0 9  | 0 7 0  | 0 2 1
0 0 0  | 4 0 9  | 0 0 0
- - - - - - - - - - - - -
0 0 1  | 0 0 0  | 8 0 0
0 7 0  | 0 4 2  | 0 0 0
5 0 0  | 8 0 0  | 0 0 7
- - - - - - - - - - - - -
4 8 0  | 1 0 0  | 0 0 4
0 0 0  | 0 0 0  | 0 0 0
0 0 9  | 6 1 3  | 0 0 0

_____________________________
0 0 0  | 0 0 0  | 0 0 9
0 0 9  | 0 7 0  | 0 2 1
0 0 0  | 4 0 9  | 0 0 0
- - - - - - - - - - - - -
0 0 1  | 0 0 0  | 8 0 0
0 7 0  | 0 4 2  | 0 0 0
5 0 0  | 8 0 0  | 0 0 7
- - - - - - - - - - - - -
4 8 0  | 1 0 0  | 0 0 4
0 0 0  | 0 0 0  | 0 0 0
0 0 9  | 6 1 3  | 0 0 0
```

Which fails in solving the problem resulting in giving the solution same as the given input
So it limits its solving capability to solving easy, medium, hard puzzles

And at the same time in this particular programme as there is no automatic puzzle detector visually giving the total 81 entries for the problem takes a bit more time but simultanously lesser than that of the time usually takes in solving one of these puzzles

9. REFERENCES :-

To conduct this project the following tools have been used :

- Jupyter notebook and spyder

1.1Geeks for Geeks

https://www-geeksforgeeks-org.cdn.ampproject.org/v/s/www.geeksforgeeks.org/backtracking-introduction/amp/?amp_js_v=a3&amp_gsa=1&usqp=mq331AQCKAE%3D#aoh=15822651568818&referrer=https%3A%2F%2Fwww.google.com&

amp_tf=From%20%251%24s&ampshare=https%3A%2F%2Fwww.geeksforgeeks.org%2Fbacktracking-introduction%2F

1.2Tech with Tim

https://techwithtim.net/tutorials/python- programming/sudoku-solver-backtracking/

1.3Wikipedia

Https://www.en.m.wikipedia.org/wiki/Backtracking

1.4Introduction to python programming by liang -text book