KNIGHT'S TOUR PROBLEM USING BACKTRACKING

Swapnil Patil
Vishwakarma Institute Of Technology
Bibwewadi, Pune 411037
Email:swapnil.patil21@vit.edu

Devyani Ushir
Vishwakarma Institute Of Technology
Bibwewadi, Pune 411037
Email:devyani.ushir 21@vit.edu

Komal Shinde
Vishwakarma Institute Of Technology
Bibwewadi, Pune 411037
Email:komal.shinde21@vit.edu

Aditya Vhanmane
Vishwakarma Institute Of Technology
Bibwewadi, Pune 411037
Email:aditya.vhanmane@vit.edu

***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, we have presented an algorithm called pencil-and-paper using human strategies. The purpose is to implement a more efficient algorithm and then compare it with another Sudoku solver named as Backtracking algorithm. This algorithm is a general algorithm that can be employed in to any problems. The results have proved that the pencil-and-paper algorithm solves the puzzle faster and more effective than the Backtracking algorithm.**

Keywords—Algorithms, pencile and paper, Backtracking

1. INTRODUCTION

Like all Other Backtracking problems, Sudoku can be solved by assigning numbers one by one to empty cells. Before assigning a number, check whether it is safe to assign. Check that the same number is not present in the current row, current column and current 3X3 sub grid. After checking for safety, assign the number, and recursively check whether this assignment leads to a solution or not. If the assignment doesn't lead to a solution, then try the next number for the current empty cell. And if none of the number (1 to 9) leads to a solution, return false and print no solution exists.

Backtracking is a kind of brute-force approach which comes into picture when solving a problem requires considering multiple choices as we don't know which choice is correct and we try to solve the problem using trail and error method considering one choice at a time until required answer is obtained.The difference between brute-force and backtracking is that in backtracking, we do not explore all the possible options instead we explore only worthy options and build the solution incrementally.

1.1 Backtracking

Backtracking is a kind of brute-force approach which comes into picture when solving a problem requires considering multiple choices as we don't know which choice is correct and we try to solve the problem using trail and error method considering one choice at a time until required answer is obtained.The difference between brute-force and backtracking is that in backtracking, we do not explore all the possible options instead we explore only worthy options and build the solution incrementally.

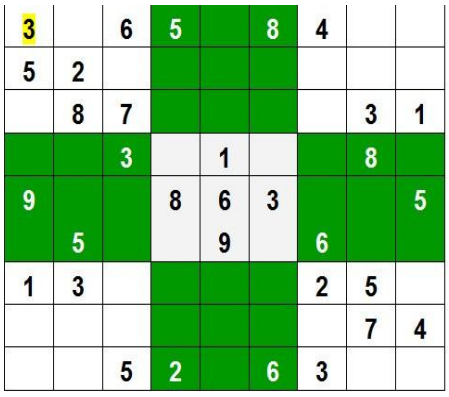


Figure 1 Sudoku Solver Problem

2. LITERATURE REVIEW

2.1 A Genetic Algorithm Based Solver For Very Large Jigsaw Puzzles[11] proposed in year 2013 has used Genetic algorithm. Genetic algorithm has set of population and population consists of chromosomes of solution possibilities. And it keeps on traversing this chromosomes until the suitable solution is found. Implementing this for Sudoku puzzle would cause the problem of slow convergence and local minima. The next literature paper i.e. Solving Sudoku using genetic algorithm[12] in year 2009 has highlighted the

flaws of using genetic algorithm for Sudoku puzzle. Further study was done on Finding solution to Sudoku puzzles using human intuitive heuristics[15] in year 2012 which also used genetic approach.

2.2 Next Study Was Done On Objects Recognition And Pose Calculation System For Mobile Augmented Reality Using Natural Features.[14] and SURFTrac: Efficient Tracking And Continuous Object Recognition Using Local Feature Descriptors[18] , year 2009 . They used Speed Up Robust Features(SURF) Algorithm which is used for continuous tracking of objects in the environment in real time. Comparing our system to this algorithm, continuous detection of object is not really needed as after the extraction of puzzle would get us the essential data for solving the puzzle.

2.3 Similar case is with Real Time Object Detection For Smart Vehicles[14] which used Distance transform simulated annealing which is detects random shapes but Hough transform in our system would detect only grid lines of the puzzle which is needed here. Further study was done on Finding solution to Sudoku puzzles using human intuitive heuristics[15] in year 2012 which also used genetic approach.

2.4 Another survey was made on Real Time Traffic Sign Recognition Based On Color Image Segmentation[16] implemented in year 2013. They have made use of joint transform correlation technique which is used for pattern matching. Similarly we use OCR(Optical Character Recognition) for recognizing the characters.
Next is Paper and Pencil approach for Sudoku puzzle solving [19] in which Combination of naked single, hidden single and backtracking is done. This is the best method for solving Sudoku puzzle as it computes the solution in short span of time.

3. ALGORITHM

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 sudoku Solver(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.
5. A sudoku solution must satisfy all of the following rules:
   5.1 Each of the digits 1-9 must occur exactly once in each row.
   5.2 Each of the digits 1-9 must occur exactly once in each column.
   5.3 Each of the digits 1-9 must occur exactly once in each of the 9 3x3 sub-boxes of the grid.
   5.4 The '.' character indicates empty cells.

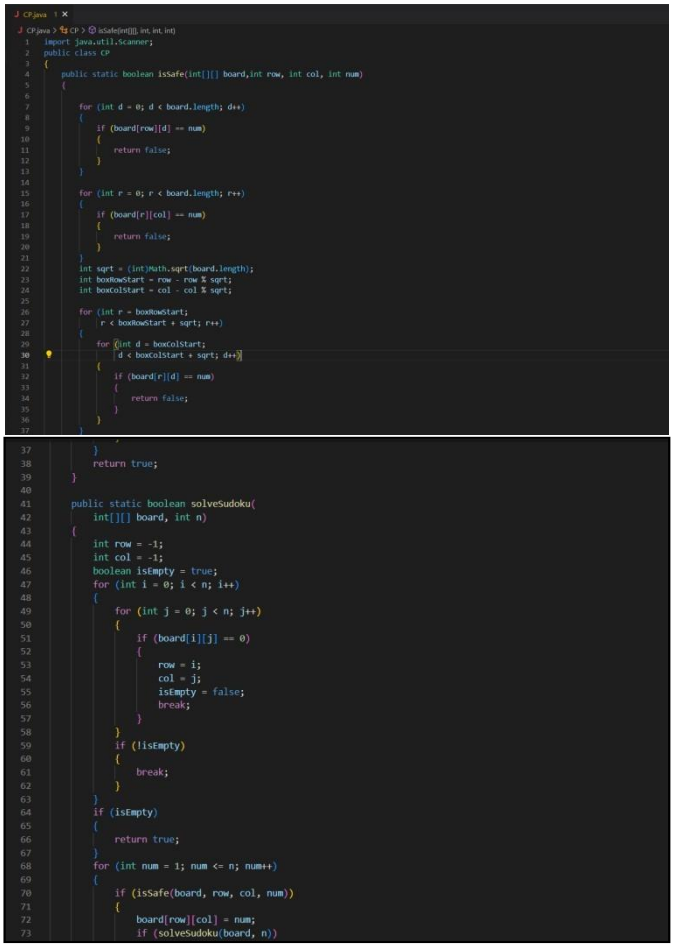
4. ANALYSIS OF CODE

The backtracking algorithm takes **$O(9^m)$** time complexity in the worst case since for every unfilled cell there are **9** possibilities to explore and there are **m** unfilled cells in the sudoku.

- Worst Case Time Complexity: $O(9^m)$
- Average Case Time Complexity: $O(9^m)$
- Best Case Time Complexity: $O(m^2)$ [This takes place when the number of backtracking steps are minimized]

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

5. IMPLEMENTATION AND RESULTS



```
CP.java
import java.util.Scanner;
public class CP
{
    public static boolean isSafe(int[][] board,int row, int col, int num)
    {

        for (int d = 0; d < board.length; d++)
        {
            if (board[row][d] == num)
            {
                return false;
            }
        }

        for (int r = 0; r < board.length; r++)
        {
            if (board[r][col] == num)
            {
                return false;
            }
        }
        int sqrt = (int)Math.sqrt(board.length);
        int boxRowStart = row - row % sqrt;
        int boxColStart = col - col % sqrt;

        for (int r = boxRowStart;
             r < boxRowStart + sqrt; r++)
        {
            for (int d = boxColStart;
                 d < boxColStart + sqrt; d++)
            {
                if (board[r][d] == num)
                {
                    return false;
                }
            }
        }
        return true;
    }

    public static boolean solveSudoku(
        int[][] board, int n)
    {
        int row = -1;
        int col = -1;
        boolean isEmpty = true;
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < n; j++)
            {
                if (board[i][j] == 0)
                {
                    row = i;
                    col = j;
                    isEmpty = false;
                    break;
                }
            }
            if (!isEmpty)
            {
                break;
            }
        }
        if (isEmpty)
        {
            return true;
        }
        for (int num = 1; num <= n; num++)
        {
            if (isSafe(board, row, col, num))
            {
                board[row][col] = num;
                if (solveSudoku(board, n))
```

```
                if (solveSudoku(board, n))
                {
                    return true;
                }
                else
                {
                    board[row][col] = 0;
                }
            }
        }
        return false;
    }

    public static void print(
        int[][] board, int N)
    {
        System.out.println(x: "Sudko Solve is :");
        System.out.println();
        for (int r = 0; r < N; r++)
        {
            for (int d = 0; d < N; d++)
            {
                System.out.print(board[r][d]);
                System.out.print(s: " ");
            }
            System.out.print(s: "\n");

            if ((r + 1) % (int)Math.sqrt(N) == 0)
            {
                System.out.print(s: "");
            }
        }
    }

    Run | Debug
    public static void main(String args[])
    {
```

```
    Run | Debug
    public static void main(String args[])
    {
        System.out.print(s: "Enter grid Size : ");
        Scanner sc=new Scanner(System.in);
        int N=sc.nextInt();

        System.out.println(x: "Enter grid elements : ");
        int[][] board = new int[N][N];

        for(int i=0; i<N;i++)
        {
            for(int j=0; j<N;j++)
            {
                board[i][j]=sc.nextInt();
            }
        }

        if (solveSudoku(board, N))
        {
            print(board, N);
        }
        else {
            System.out.println(x: "No solution");
        }
    }
}
```

Results :-

Input :-

```
Enter grid Size : 9
Enter grid elements :
3 0 6 5 0 8 4 0 0
5 2 0 0 0 0 0 0 0
0 8 7 0 0 0 0 3 1
0 0 3 0 1 0 0 8 0
9 0 0 8 6 3 0 0 5
0 5 0 0 9 0 6 0 0
1 3 0 0 0 0 2 5 0
0 0 0 0 0 0 0 7 4
0 0 5 2 0 6 3 0 0
```

Output :-

```
Sudko Solve is :

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

6. CONCLUSION

Backtracking algorithm is an appropriate method to find a solution faster and more efficient compared to the brute force algorithm. The proposed algorithm is able to solve such puzzles with any level of difficulties in a short period of time (less than one second). A Sudoku (top) being solved by backtracking. Each cell is tested for a valid number, moving "back" when there is a violation, and moving forward again until the puzzle is solved. A Sudoku designed to work against the brute force algorithm.

7. FUTURE SCOPE

Scope of the project is generating the Sudoku puzzle with different difficulty level and solves the Sudoku game. The scope covers:Allow user to select the difficulty level of the Sudoku game.Generate a complete Sudoku puzzle by following the Sudoku rules which are the digits 1-9 appear exactly once in each row, each columns, and each 3×3 sub grids.Remove cells to create a valid Sudoku games according to the difficulty level that the user selected.Allow user to input the digit 1-9 into the empty cells of the puzzle.Highlight the incorrect input of digit with red colour.Solve the Sudoku games.

REFERENCES

[1] Sholomon, D., David, O., & Netanyahu, N. S. (2013). A genetic algorithm-based solver for very large jigsaw puzzles. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1767-1774).

[2] Khan, R. U., UlHaq, M. I., Sinar, M. S., Adruce, S. A. Z., Khan, Y., & Mughal, Y. H. OBJECTS RECOGNITION AND POSE CALCULATION SYSTEM FOR MOBILE AUGMENTED REALITY USING NATURAL FEATURES.

[3] Mantere, T., & Koljonen, J. (2006, October). Solving and rating sudoku puzzles with genetic algorithms. In New Developments in Artificial Intelligence and the Semantic Web, Proceedings of the 12th Finnish Artificial Intelligence Conference STeP (pp. 86-92).

[4] Priyanka Patil , Devendra Pednekar , Tejas Modak , Sahil Kamble. Computer Vision and Artificial Intelligence Based Puzzle Solver. Volume 2, Issue 1, May 202

[5] Deshmukh, V. R., Patnaik, G. K., & Patil, M. E. (2013). Real-time traffic sign recognition system based on colour image segmentation. International Journal of Computer Applications, 83(3).

[6] Onokpasa, E., Bisandu, D., & Bakwa, D. (2019). A hybrid backtracking and pencil and paper Sudoku solver.

[7] Maji, A. K., & Pal, R. K. (2014, February). Sudoku solver using minigrid based backtracking. In 2014 IEEE International Advance Computing Conference (IACC) (pp. 36-44). IEEE.