AI (MSE 1) REPORT

TOPIC - Sudoku Solver

NAME - SATYAM TIWARI

COURSE - B-TECH

BRANCH - CSE-AI(C)

UNI. ROLL NO. – 202401100300219

CLASS ROLL NO. - 72

Introduction

Sudoku is a logic-based number puzzle game played on a 9x9 grid. The objective is to fill the grid so that each row, column, and 3x3 sub grid contains the numbers 1 to 9 without repetition. While simple Sudoku puzzles can often be solved using logical reasoning, more complex ones require systematic approaches, such as algorithmic techniques.

This project presents a Python-based Sudoku Solver that utilizes the backtracking algorithm. Backtracking is a recursive technique that explores possible number placements, checks for validity, and reverses incorrect choices when necessary. This approach ensures an efficient solution while strictly adhering to Sudoku rules.

The solver can be used to solve any valid Sudoku puzzle and can serve as a useful tool for educational purposes, puzzle verification, and automated game-solving applications.

Methodology

The Sudoku solver is implemented using a backtracking algorithm, which is a recursive technique that systematically explores all possible solutions while ensuring that the Sudoku rules are followed. The methodology consists of the following steps:

Identifying Empty Cells

The algorithm scans the 9x9 grid to locate an empty cell (represented by 0).

Trying Possible Numbers (1 to 9)

The algorithm attempts to place numbers from 1 to 9 in the empty cell.

Validation Check

Before placing a number, the is_valid() function verifies that it does not violate Sudoku rules:

The number must not be present in the same row.

The number must not be present in the same column.

The number must not be present in the corresponding 3x3 sub grid.

Recursive Solving

If a valid number is placed, the function recursively calls itself to solve the remaining empty cells.

Backtracking (Undo & Retry) :

If no valid number can be placed in a cell, the algorithm backtracks by resetting the last placed number and trying the next possibility.

This process continues until the puzzle is fully solved or determined to be unsolvable.

Solution Verification & Output

If a valid solution is found, the final solved Sudoku board is displayed.

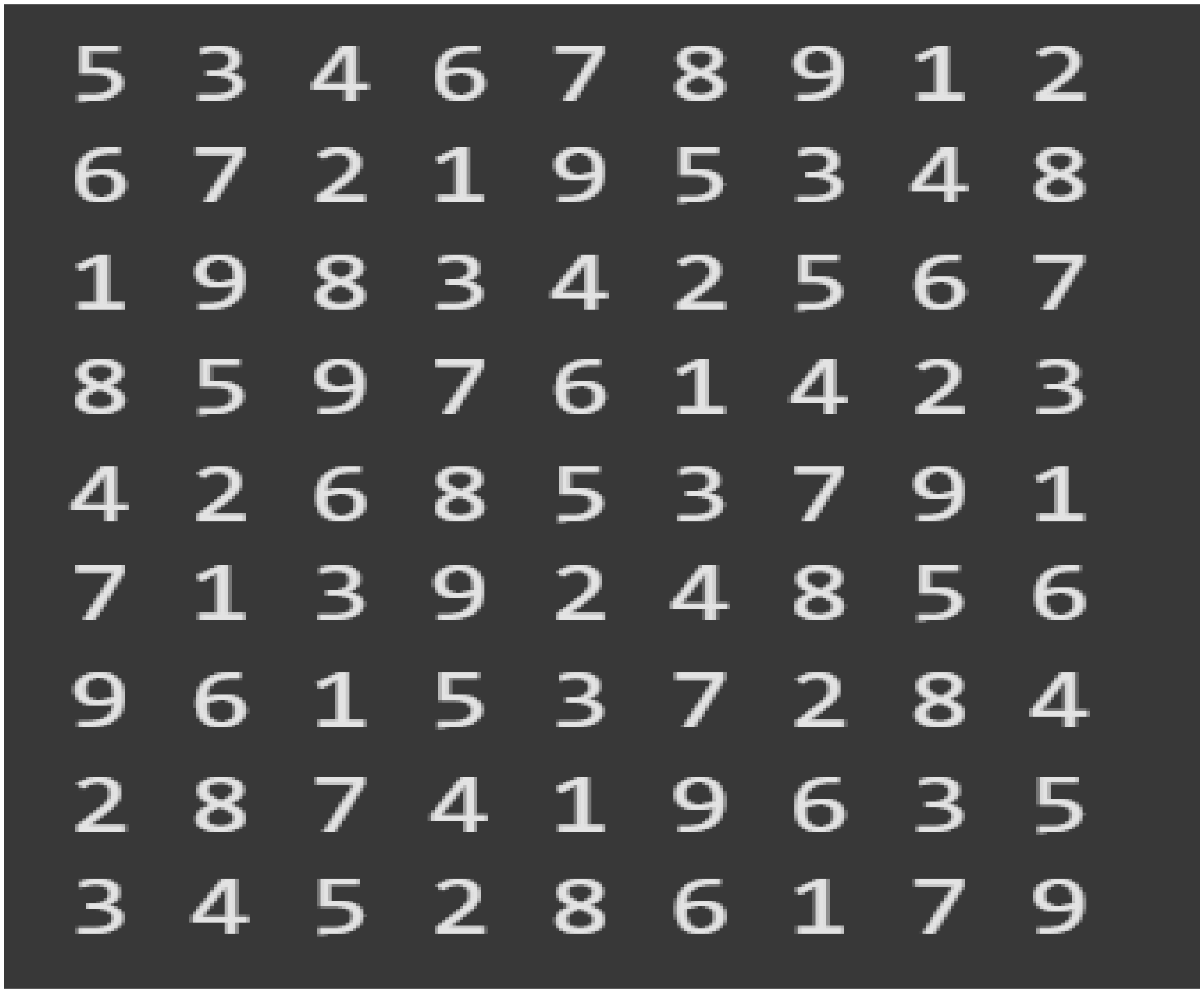
If no solution exists, an appropriate message is shown.

The backtracking approach ensures an efficient solution by eliminating incorrect possibilities early, making it one of the best techniques for solving constraint-based problems like Sudoku.

Python Code

```
def is_valid(board, row, col, num):
    """
    Checks whether placing 'num' in board[row][col] is valid.
    """
    # Check if 'num' is already in the row or column
    for i in range(9):
        if board[row][i] == num or board[i][col] == num:
            return False

    # Check if 'num' is in the 3x3 subgrid
    start_row, start_col = 3 * (row // 3), 3 * (col // 3)
    for i in range(3):
        for j in range(3):
            if board[start_row + i][start_col + j] == num:
                return False

    return True

def solve_sudoku(board):
    """
    Solves the Sudoku puzzle using backtracking.
    """
    for row in range(9):
        for col in range(9):
            # Find an empty cell (represented by 0)
            if board[row][col] == 0:
                # Try numbers 1 to 9
                for num in range(1, 10):
                    if is_valid(board, row, col, num):
                        board[row][col] = num  # Place the number

                        # Recursively attempt to solve the rest of the
board
                        if solve_sudoku(board):
                            return True

                        # If placing 'num' doesn't lead to a solution, reset
cell
                        board[row][col] = 0

                # No valid number found, trigger backtracking
                return False
    return True  # Puzzle solved

def print_board(board):
    """
    Prints the Sudoku board in a readable format.
    """
    for row in board:
        print(" ".join(str(num) if num != 0 else '.' for num in row))

# Example usage:
sudoku_board = [
    [5, 3, 0, 0, 7, 0, 0, 0, 0],
    [6, 0, 0, 1, 9, 5, 0, 0, 0],
    [0, 9, 8, 0, 0, 0, 0, 6, 0],
    [8, 0, 0, 0, 6, 0, 0, 0, 3],
    [4, 0, 0, 8, 0, 3, 0, 0, 1],
    [7, 0, 0, 0, 2, 0, 0, 0, 6],
    [0, 6, 0, 0, 0, 0, 2, 8, 0],
    [0, 0, 0, 4, 1, 9, 0, 0, 5],
    [0, 0, 0, 0, 8, 0, 0, 7, 9]
]

# Solve the puzzle and print the solution
if solve_sudoku(sudoku_board):
    print_board(sudoku_board)
else:
    print("No solution exists")
```

Screenshots of Output



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