

KIET GROUP OF INSTITUTIONS

Course name-Introduction to AI

SUDOKU SOLVER

Name-Neha yadav

Branch-CSE AI

date- 11-03-2025

Introduction

Sudoku is a popular logic-based number puzzle that requires filling a 9x9 grid with digits from 1 to 9, ensuring that no number repeats within any row, column, or 3x3 sub-grid. This report presents a Python-based Sudoku solver implemented using backtracking.

Methodology

The Sudoku solver is implemented using a backtracking algorithm:

- Identify an empty cell (denoted by 0).
- Try placing numbers from 1 to 9 in the empty cell.
- Check if the placement is valid based on Sudoku rules.
- If valid, recursively attempt to solve the rest of the puzzle.
- If no valid number fits, backtrack and try another possibility.
- Continue until the entire grid is filled correctly.

```

• import numpy as np
•
• def is_valid(board, row, col, num):
•     """Check if placing num in board[row][col] is valid."""
•     if num in board[row]:
•         return False # Check row
•     if num in [board[i][col] for i in range(9)]:
•         return False # Check column
•
•     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 # Check 3x3 grid
•
•     return True
•
• def find_empty_cell(board):
•     """Find an empty cell in the board (denoted by 0)."""
•     for i in range(9):
•         for j in range(9):
•             if board[i][j] == 0:
•                 return (i, j)
•     return None
•
• def solve_sudoku(board):
•     """Solves the Sudoku puzzle using backtracking."""
•     empty_cell = find_empty_cell(board)
•     if not empty_cell:
•         return True # No empty cell left, puzzle solved
•
•     row, col = empty_cell
•     for num in range(1, 10): # Try numbers 1-9
•         if is_valid(board, row, col, num):
•             board[row][col] = num # Place the number
•             if solve_sudoku(board):
•                 return True
•             board[row][col] = 0 # Undo move if it leads to
• failure
•
•     return False
•
• def display_board(board):
•     """Display Sudoku board in a readable format."""
•     print("\nSudoku Board:")
•     for i in range(9):

```

```

    if i % 3 == 0 and i != 0:
        print("- - - - -")
    for j in range(9):
        if j % 3 == 0 and j != 0:
            print("|", end=" ")
        print(board[i][j], end=" ")
    print()

def get_user_sudoku():
    """Get Sudoku puzzle input from the user in Google Colab."""
    board = []
    print("Enter the Sudoku puzzle row by row (use 0 for empty cells):")
    for i in range(9):
        while True:
            try:
                row = list(map(int, input(f"Row {i + 1}: ").split()))
                if len(row) == 9 and all(0 <= num <= 9 for num in row):
                    board.append(row)
                    break
            except:
                print("Invalid input. Please enter exactly 9 numbers between 0 and 9.")
        except ValueError:
            print("Invalid input. Please enter numbers only.")
    return np.array(board)

# Get user input Sudoku
sudoku_board = get_user_sudoku()
display_board(sudoku_board)

if solve_sudoku(sudoku_board):
    print("\nSolved Sudoku Board:")
    display_board(sudoku_board)
else:
    print("\nNo solution exists")

```

Solved sudoku

```
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

Solved Sudoku Board:

Sudoku Board:
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
```

Conclusion

The Sudoku solver successfully solves Sudoku puzzles using backtracking. It efficiently fills in missing numbers while ensuring Sudoku rules are followed. This approach is effective for most Sudoku puzzles and can be extended for further optimizations, such as heuristic-based solving techniques.

Future Enhancements

- Implementing graphical user interface (GUI) for better user interaction.
- Optimizing the backtracking algorithm using constraint propagation.
- Generating random Sudoku puzzles for users to solve interactively.

This project provides a foundation for automated Sudoku solving and can be further enhanced for educational and gaming applications.