Report on Sudoku Solver Using Backtracking Algorithm

Introduction

Sudoku is a widely recognized logic-based number puzzle, which requires placing digits (1 through 9) in a 9x9 grid. The puzzle has specific rules that each row, each column, and each of the nine 3x3 subgrids (also called boxes) must contain the digits 1 to 9, without repetition. Some of the grid's cells are pre-filled, while others are empty, and the challenge is to correctly fill in these empty cells following the rules.

In this report, we explore a solution to this puzzle using a **backtracking algorithm**. The backtracking approach is a brute-force method that systematically tries all possibilities to find the solution. It works by filling one cell at a time and backtracking if a conflict arises. This method is efficient and guarantees finding the solution for valid Sudoku puzzles.

Code

```
#A function to check if a number is safe to place in a given position
def is_safe(board, row, col, num):
    # Check if the number exists in the same row
    if num in board[row]:
        return False

    # Check if the number exists in the same column
    for r in range(9):
        if board[r][col] == num:
            return False

    # Check if the number exists in the same 3x3 subgrid
    start_row, start_col = 3 * (row // 3), 3 * (col // 3)
    for r in range(start_row, start_row + 3):
        for c in range(start_col, start_col + 3):
            if board[r][c] == num:
                return False

    return True
```

```
# A function to apply the backtracking algorithm
def solve(board):
    for row in range(9):
        for col in range(9):
            # If the current cell is empty (0), try to fill it with a
valid number
            if board[row][col] == 0:
                for num in range(1, 10):
                    if is_safe(board, row, col, num):  # Check if num
is valid
                        board[row][col] = num  # Place num in the cell
                        if solve(board):  # Recursively attempt to
solve
                            return True
                        board[row][col] = 0  # Undo the move
(backtrack)
                return False  # If no valid number is found, backtrack
    return True  # Puzzle solved when no empty cells remain
```

```
# Function to print the Sudoku board in a readable format
def print_board(board):
    for row in board:
        print(" ".join(str(num) if num != 0 else '.' for num in
row))  # '.' for empty cells
```

```
#Function to take input from the user for the Sudoku board
def input_board():
    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}:
").strip().split()))
                if len(row) == 9 and all(0 <= num <= 9 for num in row):
                    board.append(row)
                    break
                else:
                    print("Invalid row, please enter exactly 9 numbers
(0-9)!")
            except ValueError:
```

```
                print("Invalid input, please enter only integers!")
    return board
```

```
#Function to take input from the user for the Sudoku board
def input_board():
    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}:
").strip().split()))
                if len(row) == 9 and all(0 <= num <= 9 for num in row):
                    board.append(row)
                    break
                else:
                    print("Invalid row, please enter exactly 9 numbers
(0-9)!")
            except ValueError:
                print("Invalid input, please enter only integers!")
    return board
```

Code Output: