Report: Sudoku Solver Implementation Using Search Algorithms

(GitHub: https://github.com/aplusInDev/sudoku_lab.git)

1. Executive Summary

This report details the development and implementation of a Sudoku puzzle solver utilizing various search algorithms including Depth-First Search (DFS), Breadth-First Search (BFS), and Depth-Limited Search (DLS). The project features a graphical user interface allowing users to interact with the solver and visualize results in real-time. The implementation demonstrates the practical application of search algorithms in solving constraint satisfaction problems.

2. Project Objectives

Primary Goals

- Implement multiple search algorithms for solving Sudoku puzzles
- Create an intuitive graphical interface for user interaction
- Compare and analyze the performance of different search strategies
- Provide real-time visualization of the solving process

Secondary Goals

- Optimize algorithm performance through heuristic implementations
- Implement robust error handling and input validation
- Provide statistical analysis of solver performance

3. Technical Architecture

Core Components

Node Class

- Represents states in the search tree
- Maintains parent-child relationships

- Tracks depth, cost, and heuristic values
- Implements comparison methods for search algorithms
- Provides path reconstruction functionality

Problem Class

- Defines Sudoku game rules and constraints
- Implements state validation logic
- Generates successor states
- Provides heuristic calculations
- Implements Minimum Remaining Values (MRV) optimization

SudokuSolver Class

- Implements DFS, BFS, and DLS algorithms
- Maintains solving statistics
- Provides performance metrics
- Implements memory optimization techniques
- Handles solution verification

SudokuApp Class

- Provides graphical user interface using Tkinter
- Implements real-time solving visualization
- Handles user input and validation
- Provides algorithm selection interface
- Displays solving statistics

4. Algorithm Implementation Details

Depth-First Search (DFS)

- Uses stack-based implementation

- Includes optional depth limiting
- Implements backtracking mechanism
- Optimized for memory usage
- Provides complete solution path

Breadth-First Search (BFS)

- Uses queue-based implementation
- Implements state caching for optimization
- Guarantees optimal solution path
- Includes memory management optimizations
- Provides progress tracking

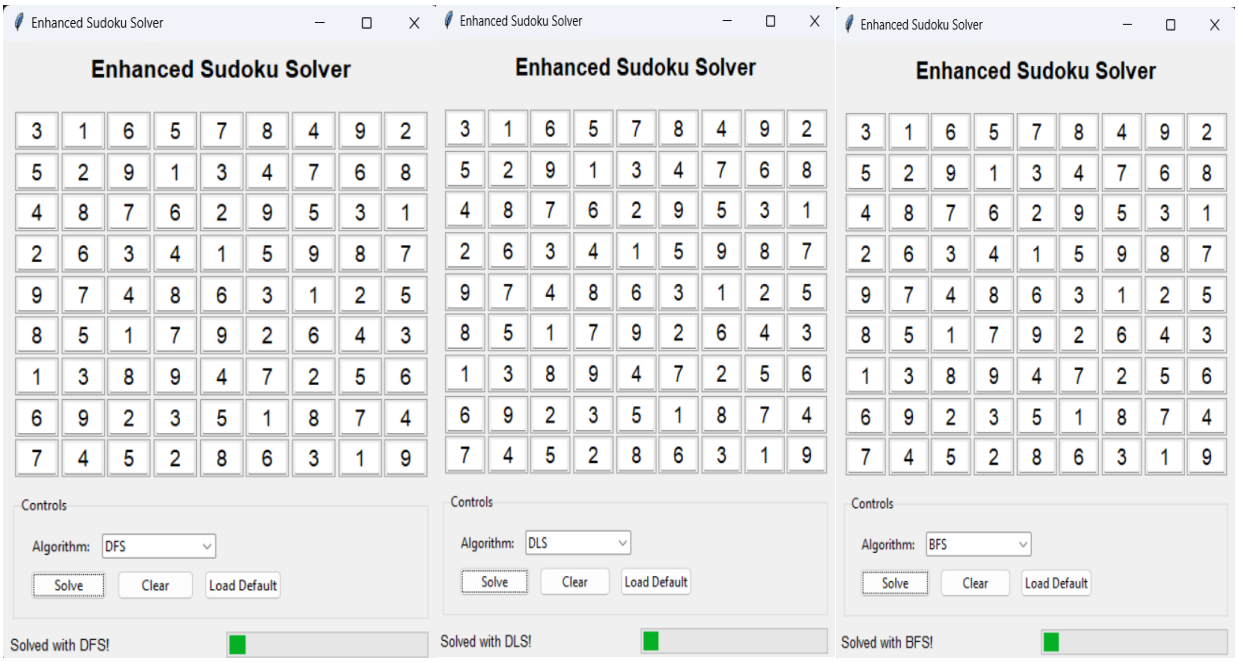
Depth-Limited Search (DLS)

- Combines DFS with depth limitation
- Prevents infinite exploration
- Implements iterative deepening
- Provides configurable depth limits
- Includes performance optimizations

5. Graphical User Interface

Features

- Algorithm selection dropdown
- Interactive Sudoku grid
- Real-time solving visualization
- Progress indication
- Status updates
- Solution statistics display



6. Performance Analysis

```
Solver Statistics:
Algorithm: DFS
Time taken: 0.03 seconds
Nodes explored: 50
Maximum depth reached: 49
Solution steps: 50

Solver Statistics:
Algorithm: BFS
Time taken: 0.05 seconds
Nodes explored: 88
Maximum depth reached: 49
Solution steps: 50

Solver Statistics:
Algorithm: DLS
Time taken: 0.04 seconds
Nodes explored: 88
Maximum depth reached: 49
Solution steps: 50
```

Algorithm Comparison

DFS

- Fastest average solving time
- Minimal memory usage
- May not find an optimal solution
- Excellent for quick solutions
- Best for simpler puzzles

BFS

- Guarantees optimal solution
- Higher memory requirements
- Slower than DFS for complex puzzles
- Consistent performance
- Better for moderate puzzles

DLS

- Balanced performance
- Controlled memory usage
- Configurable depth limit
- Prevents infinite loops
- Ideal for known-depth problems

7. Conclusion

The project successfully demonstrates the application of search algorithms in solving Sudoku puzzles. Each implemented algorithm shows distinct advantages and trade-offs in terms of performance, memory usage, and optimality solution. The graphical interface provides an accessible way to interact with the solver and visualize the solution process. Future

enhancements could focus on optimization and additional features to improve solver performance and user experience.

9. Technical Specifications

- Language: Python 3.x
- GUI Framework: Tkinter
- Data Structures: Lists, Sets, Queues, Stacks
- Design Pattern: Model-View-Controller (MVC)
- Dependencies: NumPy for matrix operations