CSE440 Section 1
Group 4
Suaiba Binte Salam 2112062642
MD Maruf Rahman 2111978642
A K Faizul Haque Konok 1931939642

Project Final Report

Project Title: Puzzle Solver

Abstract

This paper presents a multi-algorithm puzzle solver that tackles two well-known puzzles—the 8-puzzle and Sudoku—by integrating several search methods within a unified framework. The system implements Breadth-First Search (BFS), Depth-First Search (DFS) with a depth limit, and A* search (using Manhattan distance) for the 8-puzzle, as well as two backtracking-based solvers for Sudoku (a basic variant and an advanced variant using the Minimum Remaining Value heuristic). A modular design facilitates user input, algorithm selection, and performance evaluation. The system is implemented in Python and supports interactive user inputs. The output displays the solution, computational metrics such as time elapsed for both the puzzles, and the number of nodes expanded for the 8-Puzzle. This report elaborates on the design, algorithms, implementation, and usage of the Puzzle Solver.

Keywords

8-Puzzle, Sudoku, Breadth-First Search, Depth-First Search, A* Search, Backtracking, MRV heuristic, puzzle solve.

Introduction

Puzzle-solving is a classic research problem in artificial intelligence and computer science. The 8-Puzzle and Sudoku are widely studied puzzles that present unique challenges in terms of search space and constraints. The primary goal of this project is to develop a unified puzzle solver that integrates multiple algorithms to efficiently solve the 8-puzzle and Sudoku. The implementation emphasizes flexible algorithm selection, custom user input, and performance evaluation. It connects theoretical algorithm design to practical problem-solving.

CSE440 Section 1
Group 4
Suaiba Binte Salam 2112062642
MD Maruf Rahman 2111978642
A K Faizul Haque Konok 1931939642

Problem Statement

The two puzzles addressed in this project are:

- **8-Puzzle:** A 3x3 sliding puzzle with eight numbered tiles and one blank space (0 representing the blank space). The objective is to move the blank tile and rearrange them to reach a goal state.
- **Sudoku:** A logic-based number-placement puzzle. It consists of 9x9 grid, that is subdivided into 3x3 subgrids. The aim is to fill the empty spaces in such a way so that each row, column, and subgrid contains all digits from 1 to 9 with no repetition.

These puzzles require different algorithms due to their constraints and search spaces. The 8-puzzle is typically solved using search algorithms, and Sudoku is more efficiently solved using constraint propagation and backtracking.

Related Work

The 8-Puzzle has long been used in research, with studies focusing on search methods (BFS, DFS) and informed search methods (A* search). A* remains a gold standard due to its optimality and performance when coupled with admissible heuristics like Manhattan distance. For Sudoku, backtracking has been the predominant algorithm due to the puzzle's well-defined constraints, with enhancements such as the Minimum Remaining Value (MRV) heuristic to improve efficiency. Our work builds on these foundations by integrating both puzzle problems into a single and modular framework that allows flexible selection of both problem and solution method.

System Design and Methodology

A. Modular Architecture

The system is designed with a modular architecture that separates functionalities into two distinct modules:

- **8-Puzzle Module:** Implements a class-based design where the puzzle is represented as a 3×3 grid. This module includes functions to generate possible moves (neighbors) and employs BFS, DFS (with a maximum depth parameter), and A* search (using Manhattan distance) to explore the state space.
- **Sudoku Module:** Provides two variants of a recursive backtracking algorithm. One variant uses basic backtracking, while the other implements an

advanced strategy using the Minimum Remaining Value (MRV) heuristic to select the next empty cell with the fewest candidate numbers.

B. User Input and Algorithm Selection

A key design decision was to allow users to define the puzzle configuration.

- For the 8-puzzle, the user is prompted to input 9 space-separated numbers (0 representing the blank tile) that are then converted into a 3×3 grid.
- For Sudoku, the user provides 9 rows of 9 numbers each, with zeros indicating empty cells.

Users are also given the option to choose the algorithm for solving each puzzle, which helps them in having direct performance comparisons and encourages experimental analysis.

C. Performance Metrics

To evaluate the effectiveness of each algorithm:

- For the 8-puzzle, performance is measured by counting the number of nodes expanded during search and the total execution time.
- For Sudoku, execution time is recorded for both the basic and advanced backtracking approaches.

Implementation

The system is implemented in Python with a modular structure for scalability. The code consists of three main components:

1. **8-Puzzle Solver:**
    - State Representation: A 3x3 matrix implemented as a list of lists.
    - Search Algorithms: BFS, DFS (with maximum depth limit check), and A* search (using Manhattan distance).
    - User Input Function: Validated the input provided by the user to ensure a correct puzzle configuration.
2. **Sudoku Solver:**
    - Basic Solver: Integrates recursive backtracking to assign values to empty spaces maintaining the condition of no repetition of values.
    - Advanced Solver: Uses the Minimum Remaining Value heuristic to fill the empty spaces with the least number of possible candidate values, which helps in reducing search complexity.
    - Input and Validation: Prompts the user to

- provide 9 rows of input and checks the data format.

3. **Main Program:**
   - Displays a menu for the user to choose the puzzle type, algorithm variant and asks for inputs.
   - Outputs the solution along with the performance metrics for comparison.

Input Example:

- **8-Puzzle**: `1 2 3 4 5 6 7 8 0`
- **Sudoku**: 9 rows with 9 values each (0 for empty cells)

  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

Experimental Results

During testing, in the 8-puzzle solver using A* search consistently outperformed the BFS and DFS implementations in terms of nodes expanded and time taken. DFS having a limit set for its maximum depth search, occasionally failed when the solution path exceeded that limit. In case of Sudoku, the advanced backtracking approach with MRV

showed better results than the basic backtracking method.

Execution Flow

After running the program, the user is prompted to choose the puzzle type and inputs. If the provided input is correct, the program than asks the user to choose an algorithm for solving the puzzle. After that, the program executes using the chosen method and returns the results, including the time taken and, in the case of the 8-Puzzle, the number of nodes expanded.

Input and Output for 8-Puzzle:

```
Current 8-Puzzle State:
4 2 8
1   3
5 7 6

Select search algorithm for the 8-Puzzle Solver:
1. Breadth-First Search (BFS)
2. Depth-First Search (DFS)
3. A* Search
Enter your choice (1/2/3): 2

Algorithm: DFS
Solution found in 98 moves: ['Right', 'Down', 'Left',
Nodes expanded: 162635
Time taken: 0.4178 seconds
```

```
Current 8-Puzzle State:
4 2 8
1   3
5 7 6

Select search algorithm for the 8-Puzzle Solver:
1. Breadth-First Search (BFS)
2. Depth-First Search (DFS)
3. A* Search
Enter your choice (1/2/3): 3

Algorithm: A* Search
Solution found in 18 moves: ['Left', 'Down', 'Right'
Nodes expanded: 587
Time taken: 0.0132 seconds
```

Input and Output for Sudoku:

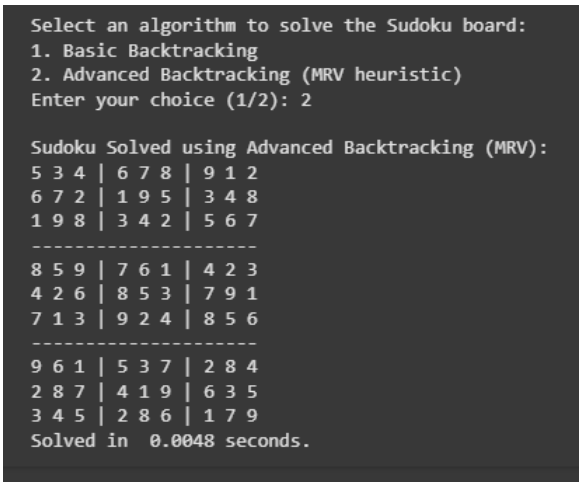
```
Puzzle Solver Menu:
1. 8-Puzzle Solver
2. Sudoku Solver
Enter your choice (1/2): 2
Enter the Sudoku puzzle row by row.
Use 0 for empty cells. Enter 9 numbers separated by spaces per row.
Row 1: 5 3 0 0 7 0 0 0 0
Row 2: 6 0 0 1 9 5 0 0 0
Row 3: 0 9 8 0 0 0 0 6 0
Row 4: 8 0 0 0 6 0 0 0 3
Row 5: 4 0 0 8 0 3 0 0 1
Row 6: 7 0 0 0 2 0 0 0 6
Row 7: 0 6 0 0 0 0 2 8 0
Row 8: 0 0 0 4 1 9 0 0 5
Row 9: 0 0 0 0 8 0 0 7 9

Initial Sudoku Board:
5 3 . | . 7 . | . . .
6 . . | 1 9 5 | . . .
. 9 8 | . . . | . 6 .
---------------------
8 . . | . 6 . | . . 3
4 . . | 8 . 3 | . . 1
7 . . | . 2 . | . . 6
---------------------
. 6 . | . . . | 2 8 .
. . . | 4 1 9 | . . 5
. . . | . 8 . | . 7 9
```

```
Select an algorithm to solve the Sudoku board:
1. Basic Backtracking
2. Advanced Backtracking (MRV heuristic)
Enter your choice (1/2): 1

Sudoku Solved using Basic Backtracking:
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
Solved in  0.0636 seconds.
```

CSE440 Section 1
Group 4
Suaiba Binte Salam 2112062642
MD Maruf Rahman 2111978642
A K Faizul Haque Konok 1931939642



Discussion

The experimental results confirmed that the heuristic-based approaches can significantly enhance search efficiency. For 8-Puzzle, the A* search with Manhattan distance and for Sudoku, the advanced backtracking using MRV heuristic gave the best performances. Moreover, the modular design allows users to make direct comparison of performances for different algorithms under similar constraints.

Challenges

- **Integration of Multiple Puzzle Types:** Combining two completely different puzzles within a unified framework required a careful modular design. We had to change the design quite a number of times to fit the requirements for the project.
- **Algorithm Selection:** To ensure that the users can easily select between multiple algorithms was a challenge for us to validate the user inputs and performance measurement.
- **Depth Limitation in DFS:** The DFS algorithm was taking too much space due to its depth exploration feature. We had to set an appropriate maximum depth limit for the algorithm to prevent the algorithm from exploring excessively deep search trees. Which resolved the space and memory issue.
- **User Input Validation:** Thoroughly validating user input to maintain data integrity was essential to get the correct evaluation of the results. Particularly for Sudoku where the user had to input 9 rows of input that are correctly parsed.

Conclusion

This report presented the development of a multi-algorithm puzzle solver that integrates solution for both the 8-Puzzle and Sudoku. The modular design ensures that the code is easy to maintain and expand in the future. It

also allowed the user to choose algorithms and input their own puzzle configurations, with performance metrices highlighting the efficiency of heuristic approaches. This project meets the project's initial goals and provides a robust foundation for further exploration in puzzle-solving algorithms.

References

1. GeeksforGeeks. "8 Puzzle Problem Using Branch and Bound." *GeeksforGeeks*, 23 Feb. 2025, www.geeksforgeeks.org/8-puzzle-problem-using-branch-and-bound/.
2. Suryadevara, Chandra. "8 Puzzle Problem." *GitHub Repository*, https://github.com/Chandra-Suryadevara/8-Puzzle-Problem.
3. GeeksforGeeks. "Sudoku Backtracking – A Recursive Backtracking Algorithm to Solve Sudoku." *GeeksforGeeks*, 23 Feb. 2025, www.geeksforgeeks.org/sudoku-backtracking-7/.
4. TutorialsPoint. "Functional Programming – Tuple." *TutorialsPoint*, www.tutorialspoint.com/functional_programming/functional_programming_tuple.htm.
5. W3Schools. "Python Tuples." *W3Schools*, www.w3schools.com/python/python_tuples.asp.
6. TutorialsPoint. "Functional Programming – Tuple." *TutorialsPoint*, www.tutorialspoint.com/functional_programming/functional_programming_tuple.htm.
7. Tech With Tim. "Python Recursion Tutorial – How Recursion Works." *YouTube*, uploaded by Tech with Tim, 20 Aug. 2018, https://www.youtube.com/watch?v=eqUwSA0xI-s.
8. Tech With Tim. "Python Recursion Tutorial – How Recursion Works." *YouTube*, uploaded by Tech with Tim, 20 Aug. 2018, https://www.youtube.com/watch?v=lK4N8E6uNr4&t=7s.
9. "ChatGPT." *ChatGPT*, https://chatgpt.com/.
10. "A* Search Algorithm to Move from Start State to Final State – 8 Puzzle Problem." *YouTube*, uploaded by Dr. Mahesh H, 3.4 years ago, https://www.youtube.com/watch?v=dvWk0vgHijs.

11. “Try and Except Blocks in Python (Error Handling).” *YouTube*, uploaded by StudySession, 5 Apr. 2021, https://www.youtube.com/watch?v=JjOUYM1xnjQ.
12. Python Software Foundation. "Built-in Types — Tuples." *Python Documentation*, https://docs.python.org/3/library/stdtypes.html#tuple.