Group 4

Course: CSE440 Section:01

Habib Foysal ID:2122284642 Emrose Khan ID: 2011932642

Md Nazmus Saqib ID:2021696642

Ashikur Rahman Ashik ID: 2111632642 Masud Shahriar ID: 2021850642

Project Final Report

Project Title: Puzzle Solver

Abstract

This report will provide an overview of a multi-algorithm puzzle solver that works for two well-known problems, the 8-puzzle and Sudoku, using many different methodologies in a single system. The 8-puzzle solver implements Breadth-First Search (BFS), Depth-First Search (DFS) with a depth limit, and A* search with a heuristic of Manhattan distance. The Sudoku solver uses two implicit backtracking methods, a basic solver and an enhanced solver with Minimum Remaining Value (MRV) as an additional heuristic. The system is designed using a modular architecture that allows for user input, algorithm choice, and performance comparison. The project was developed in Python, and can accept interactive inputs, with outputs that include the solution, total time for both puzzle types, and the total number of nodes expanded for the 8-puzzle. The body of this report describes the system design, algorithm choices, implementation and real-world usability in detail.

Keywords

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

Introduction

Puzzle-solving has been a fundamental area of study in both artificial intelligence and computer science for many years. Some areas of puzzle-solving that have been studied extensively by the scientific community include the 8-Puzzle and Sudoku, although they each present challenges in terms of search space and constraints. The main objective of this project is to create a unified solver that combines multiple algorithms to effectively handle both puzzles. The system focuses on flexible algorithm selection, user- defined inputs, and performance assessment, thereby bridging theoretical algorithm design with real-world problem-solving.

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.

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.

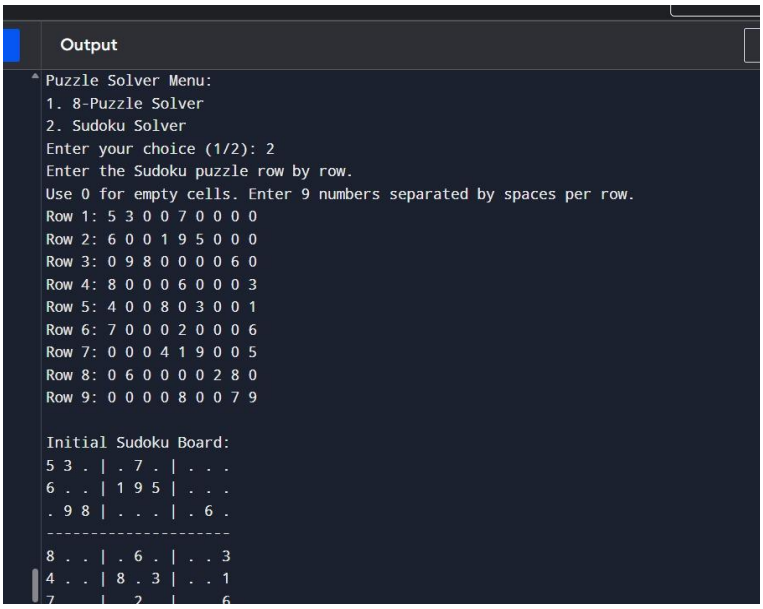
    - Input and Validation: Prompts the user to

- 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)



Experimental Results

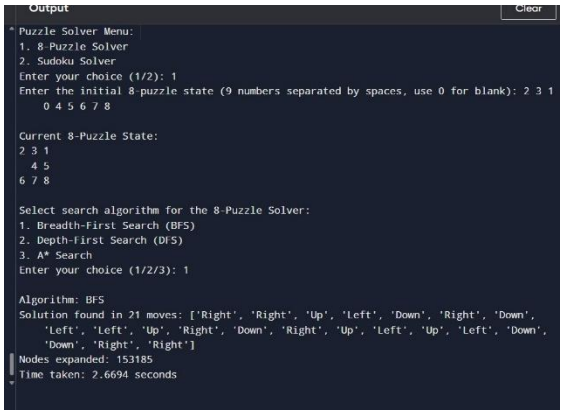
During testing, in the 8-puzzle solver using A* search consistently outperformed the BFS and DFS implementations in terms of nodes expanded and time taken. DFS having a limit set for its maximum depth search, occasionally failed when the solution path exceeded that limit. In case of Sudoku, the advanced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

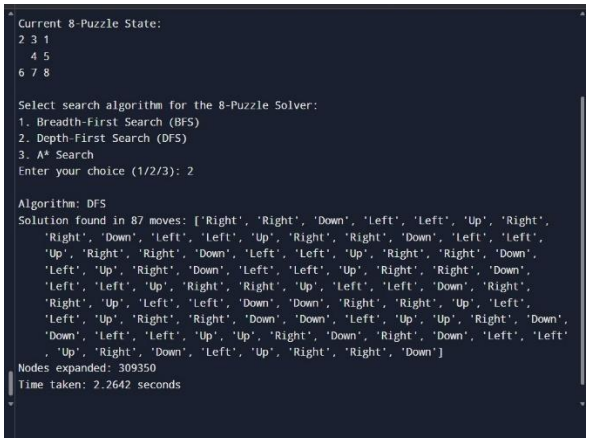
showed better results than the basic backtracking method.

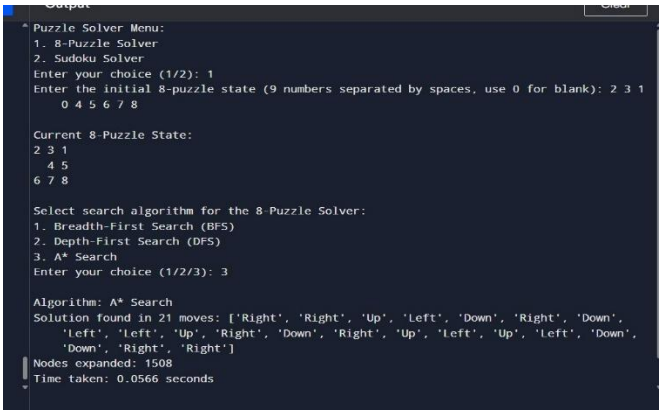
Execution Flow

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.

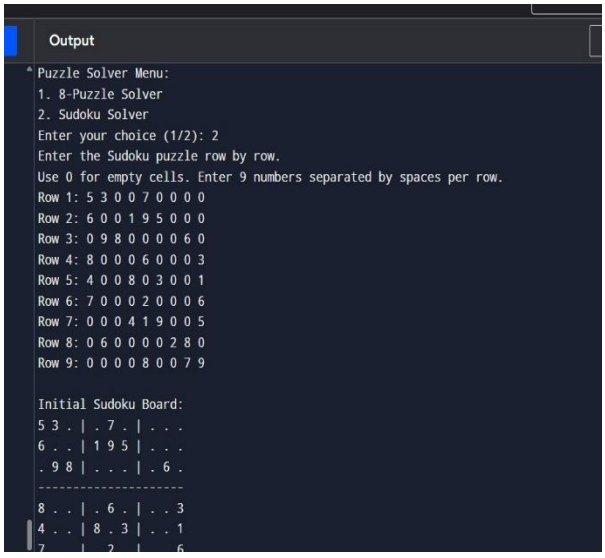
Input and Output for Puzzle

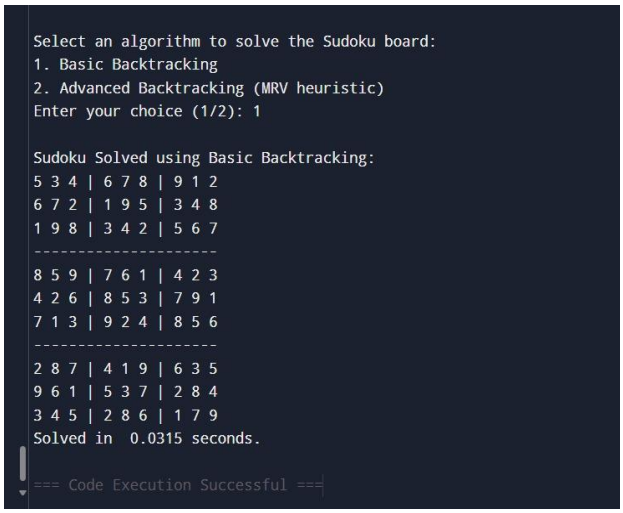


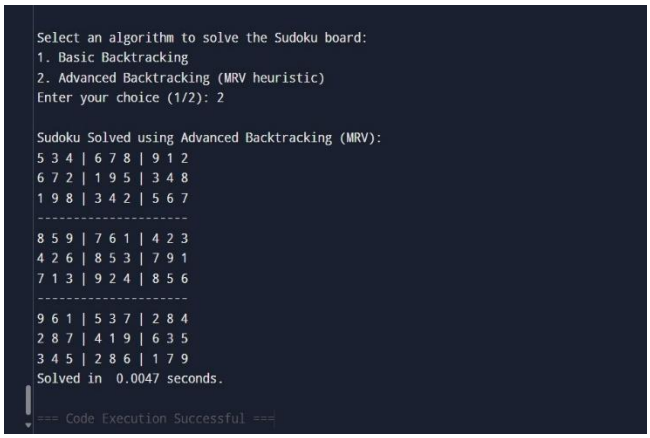




Input and Output for Sudoko







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 describes the development of a multi-algorithm puzzle solver that can use the 8-Puzzle and Sudoku. Due to the modular structure of the system, it is more manageable to sustain and extend in later developments. Further, the design allows users to select algorithms and provide their own

puzzle inputs, and the performance measurements. Overall, the project successfully fulfills its original objectives and establishes a strong basis for future research and advancements in puzzle-solving algorithms

References

G. B. Balogun, D. Ibisagba, A. Bajeh, T. O. Debo, A. Muyideen, and O. J. Peter, "Comparative analysis of AI-based search algorithms in solving 8 puzzle problems," *Bulletin of the National Research Centre*, vol. 48, no. 1, Nov. 2024,doi:https://doi.org/10.1186/s42269-024-01274-3.

w3Schools, "Python Tuples," *www.w3schools.com*, 2024. https://www.w3schools.com/python/python_tuples.asp

R. Kumar, "The A* Algorithm: A Complete Guide," *Datacamp.com*, Nov. 07, 2024. https://www.datacamp.com/tutorial/a-star-algorithm

T. Kumar, "Sudoku- Backtracking algorithm and visualization," *Analytics Vidhya*, May 04, 2021. https://medium.com/analytics-vidhya/sudoku-backtracking-algorithm-and-visualization-75adec8e860c

GeeksforGeeks, "Algorithm to Solve Sudoku | Sudoku Solver," *GeeksforGeeks*, Jul. 14, 2012.https://www.geeksforgeeks.org/dsa/sudoku-backtracking-7/

Telusko, "#0 Python for Beginners | Programming Tutorial," *YouTube*, Aug. 16, 2018.https://www.youtube.com/watch?v=QXeEoD0pB3E&list=PLsyeobzWxl7poL9JTVyndKe62ieoN-MZ3&index=2 (accessed Aug. 17, 2025).

M. Breuss, "Build Your Python Project Documentation With MkDocs," *Realpython.com*, Jun. 15, 2022. https://realpython.com/python-project-documentation-with-mkdocs/ (accessed Aug. 17, 2025).