CS205 Project 1 The 8 Puzzles

Hugo Wan, 862180666, twan012

Due Date: May 8th, 2025

Cover Page

Outline of this report:

1 Introduction

This is a report that summarizes my findings in solving a eight-puzzles using three different algorithms. My choice of coding language: C++:

1. Uniform Cost Search (UCS)
2. A* with the Misplaced Tile heuristic (AMT)
3. A* with the Manhattan Distance heuristic (AMD)

Link to play test my codes:

`https://github.com/HugoWan0504/CS205_Project_1_The_8_Puzzles`

2 The Eight Puzzle

The eight puzzle is a classic 3×3 sliding puzzle that consists of eight numbered tiles (1 through 8) and one empty space, represented by the digit `0`. The tiles can be moved into the empty space to reorder the puzzle with the goal of reaching a specific configuration.

```
Goal State:      A valid puzzle must:
[1 2 3]            1. Contain all digits from 0 to 8 exactly once
[4 5 6]            2. Be arranged in a 3×3 grid format
[7 8 0]
                    Players are allowed to use either a default
                  puzzle or enter a custom configuration, as long as
                  it follows the validity rules listed above.
```

3 Uniform Cost Search (Uninformed)

Uniform Cost Search (UCS) performs an uninformed search based on the lowest cost of the path, reaching the goal with a minimum cost since the initial state. [1] According to the lecture slide in `2_Blind Search Part 2` [2], UCS trees expand the cheapest `g(n)` node. It is a search algorithm that is considered complete and optimal when increasing in depth. It has a time and space complexity of $O(b^d)$. In the notes of the same slide, it is mentioned that UCS's special case is Breadth First search (BFS), where the path cost is equivalent to the depth of the tree.

4 Heuristic Search & The A* Algorithm

Before introducing the next A* search algorithms, heuristic search and the A* algorithm are the fundamental of the algorithms:

4.1 Heuristic Search

According to the lecture file "3_Heuristic Search" slide 24 [5], a heuristic is a function that estimates the merit or the distances of the state to its goal. It has a notation of `h(n)`. Similarly to UCS's `g(n)`, the smaller the number in `h(n)`, the cheaper the cost of the path.

4.2 The A* Algorithm

In slide 35, A* is considered optimal and complete. [6] It is annotated as `f(n)`, where it estimates the cheapest cost solution to the goal. You can say that A* is a combination of uniform cost search and heuristic search. Recall that `g(n)` is the cost to a node and `h(n)` is the estimated distance to the goal. The A* evaluation function is:

$$f(n) = g(n) + h(n)$$

In this project, two different heuristics are used to guide A*:

- Misplaced Tile: Counts the number of tiles out of place
- Manhattan Distance: Sums the row and column distances of each tile to its goal position

Both costs of h(n) do not include the calculation of the blank tile. [3, 4]

5 Conclusion

This project helped me understand search algorithms more deeply by actually building and testing them on the 8-puzzle. I implemented Uniform Cost Search (UCS), A* with the Misplaced Tile heuristic (AMT), and A* with the Manhattan Distance heuristic (AMD), and then compared how they behaved on both simple and difficult puzzles.

I started with a default puzzle that required only two moves to solve. All three algorithms reached the goal correctly, but I noticed that UCS expanded 9 nodes, while AMT and AMD only needed 3. That already showed me how much more efficient the heuristic-based methods can be.

Then I tested a custom puzzle that had a solution depth of 14. This time, the differences were way more obvious: UCS expanded 4210 nodes, AMT needed 245, and AMD only expanded 90. It was really satisfying to see how the better heuristic made such a big impact. AMD not only found the same optimal path, but did so with far less searching.

Writing the code in a modular way also helped me a lot. I kept one general solve() function that worked with all three strategies, and it made debugging and testing easier. I also made sure to support both default and custom inputs, which let me try lots of different cases.

Overall, seeing the g(n), h(n), and f(n) values change at every step, and watching the traceback unfold, really helped me connect the theory to the actual process. I feel like I learned not just how these algorithms work, but also why some of them are more practical than others.

6 User Interaction & Input Handling

The program starts by asking the user whether to use a default puzzle or enter a custom configuration. For custom input, the user is asked to enter three rows, each containing three space-separated numbers.

If the input fails validation (e.g., missing numbers or duplicates), the program rejects the puzzle and restarts the input process. Once a valid puzzle is entered, the user selects one of the following search strategies:

1. Uniform Cost Search (UCS)
2. A* with the Misplaced Tile heuristic

3. A* with the Manhattan Distance heuristic

As the algorithm runs, the program displays the current state that is expanding, along with its values g(n), h(n), and f(n). After reaching the goal, the following results are printed:

- Final puzzle state
- Number of nodes expanded
- Maximum queue size encountered
- Solution depth

Lastly, the user is then asked whether they would like to solve another puzzle or exit the program.

7 Coding Page

Link to play test my codes:

`https://github.com/HugoWan0504/CS205_Project_1_The_8_Puzzles`

Files and Their Functionalities:

1. main.cpp: Handles user interaction and terminal interface. Displays the welcome menu, lets users choose between a default or custom puzzle, selects a solving algorithm, and then runs the solver. In addition, it prompts the user to play again or exit.
2. puzzle.h/puzzle.cpp: Defines and implements the Puzzle class. Can load a default puzzle or accepts and validates user inputs. Ensures that the input contains all digits from 0 to 8 without duplicates. Stores the current puzzle state as a 3x3 grid. Prints the current puzzle to the terminal.
3. solver.h/solver.cpp: Implements the main logic for solving the 8-puzzle. Contains the generalized search algorithm that handles all three search algorithms. Moreover, it tracks and prints the number of the nodes expanded, the depth of the solutions, the size of the maximum queue. Outputs the best node expanded at each step for traceability.
4. heuristic.h/heuristic.cpp: Defines the two heuristic functions used in A* in which counting how many tiles are in the wrong position (excluding the blank) in the Misplaced Tile heuristic and Sums the distances each tile is away from its goal position (row + col moves) in the Manhattan Distance heuristic.
5. utils.h: Includes small helper functions, such as parsing a string of user input into individual numbers or checking whether a set of numbers contains all digits from 0 to 8 exactly once.

8 Default Traceback

The following texts are the traceback of the **default** puzzles for all three search algorithms:

```
Welcome to the 8-Puzzle Solver!
Choose an option:
1. Use default puzzle
2. Enter your own puzzle
Enter 1 or 2: 1
Default puzzle loaded.

Puzzle state:
1 2 3
4 0 6
7 5 8


Choose a search algorithm:
1. Uniform Cost Search (UCS)
2. A* with Misplaced Tile heuristic (AMT)
3. A* with Manhattan Distance heuristic (AMD)
Enter 1, 2, or 3: 1
Expanding node with g(n) = 0, h(n) = 0, f(n) = 0:
1 2 3
4 0 6
7 5 8

....   // Delete 7 more expanding nodes

Expanding node with g(n) = 2, h(n) = 0, f(n) = 2:
1 2 3
4 5 6
7 8 0

Goal state reached!

Step 0: g(n)=0, h(n)=0, f(n)=0
1 2 3
4 0 6
7 5 8

Step 1: g(n)=1, h(n)=0, f(n)=1
1 2 3
4 5 6
7 0 8
```

```
Step 2: g(n)=2, h(n)=0, f(n)=2
1 2 3
4 5 6
7 8 0

Solution depth: 2
Nodes expanded: 9
Max queue size: 7

Would you like to play again? (Y/y = Yes, N/n = No): y
---------------------------------------------
Welcome to the 8-Puzzle Solver!
Choose an option:
1. Use default puzzle
2. Enter your own puzzle
Enter 1 or 2: 1
Default puzzle loaded.

Puzzle state:
1 2 3
4 0 6
7 5 8

Choose a search algorithm:
1. Uniform Cost Search (UCS)
2. A* with Misplaced Tile heuristic (AMT)
3. A* with Manhattan Distance heuristic (AMD)
Enter 1, 2, or 3: 2

.... // Delete the 3 expanding nodes

Goal state reached!

Step 0: g(n)=0, h(n)=2, f(n)=2
1 2 3
4 0 6
7 5 8

Step 1: g(n)=1, h(n)=1, f(n)=2
1 2 3
4 5 6
7 0 8

Step 2: g(n)=2, h(n)=0, f(n)=2
1 2 3
```

```
4 5 6
7 8 0

Solution depth: 2
Nodes expanded: 3
Max queue size: 4

Would you like to play again? (Y/y = Yes, N/n = No): y
---------------------------------------------
Welcome to the 8-Puzzle Solver!
Choose an option:
1. Use default puzzle
2. Enter your own puzzle
Enter 1 or 2: 1
Default puzzle loaded.

Puzzle state:
1 2 3
4 0 6
7 5 8


Choose a search algorithm:
1. Uniform Cost Search (UCS)
2. A* with Misplaced Tile heuristic (AMT)
3. A* with Manhattan Distance heuristic (AMD)
Enter 1, 2, or 3: 3

.... // Delete the 3 expanding nodes

Goal state reached!

Step 0: g(n)=0, h(n)=2, f(n)=2
1 2 3
4 0 6
7 5 8

Step 1: g(n)=1, h(n)=1, f(n)=2
1 2 3
4 5 6
7 0 8

Step 2: g(n)=2, h(n)=0, f(n)=2
1 2 3
4 5 6
7 8 0
```

```
Solution depth: 2
Nodes expanded: 3
Max queue size: 4

Would you like to play again? (Y/y = Yes, N/n = No): n
---------------------------------------------
Thanks for playing. Exiting now.
```

9 Custom Traceback

The following texts are the traceback of the **custom** puzzles for all three search algorithms:

```
Welcome to the 8-Puzzle Solver!
Choose an option:
1. Use default puzzle
2. Enter your own puzzle
Enter 1 or 2: 2
Enter your puzzle row-by-row (use 0 for blank):
Row 1: 1 2 3
Row 2: 8 7 6
Row 3: 5 4 0

Puzzle state:
1 2 3
8 7 6
5 4 0

Choose a search algorithm:
1. Uniform Cost Search (UCS)
2. A* with Misplaced Tile heuristic (AMT)
3. A* with Manhattan Distance heuristic (AMD)
Enter 1, 2, or 3: 1

.... // Delete the expanding nodes up to g(n) = 14

Expanding node with g(n) = 14, h(n) = 0, f(n) = 14:
1 2 3
4 5 6
7 8 0

Goal state reached!

Step 0: g(n)=0, h(n)=0, f(n)=0
1 2 3
```

```
8 7 6
5 4 0

Step 1: g(n)=1, h(n)=0, f(n)=1
1 2 3
8 7 0
5 4 6

.... // Delete the UCS optimal steps up to step 12

Step 13: g(n)=13, h(n)=0, f(n)=13
1 2 3
4 5 6
7 0 8

Step 14: g(n)=14, h(n)=0, f(n)=14
1 2 3
4 5 6
7 8 0

Solution depth: 14
Nodes expanded: 4210
Max queue size: 2512
Would you like to play again? (Y/y = Yes, N/n = No): y
---------------------------------------------
Welcome to the 8-Puzzle Solver!
Choose an option:
1. Use default puzzle
2. Enter your own puzzle
Enter 1 or 2: 2
Enter your puzzle row-by-row (use 0 for blank):
Row 1: 1 2 3
Row 2: 8 7 6
Row 3: 5 4 0

Puzzle state:
1 2 3
8 7 6
5 4 0

Choose a search algorithm:
1. Uniform Cost Search (UCS)
2. A* with Misplaced Tile heuristic (AMT)
3. A* with Manhattan Distance heuristic (AMD)
Enter 1, 2, or 3: 2
```

```
.... // Similar to the traceback above, delete most
.... // of the output nodes for report readability
.... // Delete the expanding nodes up to g(n) = 14
.... // Delete the UCS optimal steps up to step 13

Step 14: g(n)=14, h(n)=0, f(n)=14
1 2 3
4 5 6
7 8 0

Solution depth: 14
Nodes expanded: 245
Max queue size: 170

.... // This time solving the same custom puzzle
.... // with AMD search algorithm

.... // Similar to the traceback above, delete most
.... // of the output nodes for report readability

.... // Delete the expanding nodes up to g(n) = 14
.... // Delete the UCS optimal steps up to step 13

Step 14: g(n)=14, h(n)=0, f(n)=14
1 2 3
4 5 6
7 8 0

Solution depth: 14
Nodes expanded: 90
Max queue size: 59
```

References

[1] www.educative.io https://www.educative.io/answers/what-is-uniform-cost-search Paragraph 3.

[2] Dr Eammon Keogh 2_Blind Search Part 2 PPT Slide 16.

[3] Dr Eammon Keogh 3_Heuristic Search PPT Slide 25.

[4] Dr Eammon Keogh 3_Heuristic Search PPT Slide 26.

[5] Dr Eammon Keogh 3_Heuristic Search PPT Slide 24.

[6] Dr Eammon Keogh 3_Heuristic Search PPT Slide 35.