Report on the 8-Puzzle Problem

Introduction

The 8-puzzle, also known as the sliding puzzle, is a classic problem in the field of artificial intelligence and computer science. It serves as an illustrative example of a search problem and is often used to demonstrate various search algorithms and heuristics. The 8-puzzle is a type of sliding puzzle that consists of a 3x3 grid with eight numbered tiles and an empty space. The objective is to rearrange the tiles from their initial, often scrambled, state into a goal state, which is usually in ascending order from left to right, top to bottom, with the empty space in the bottom-right corner.

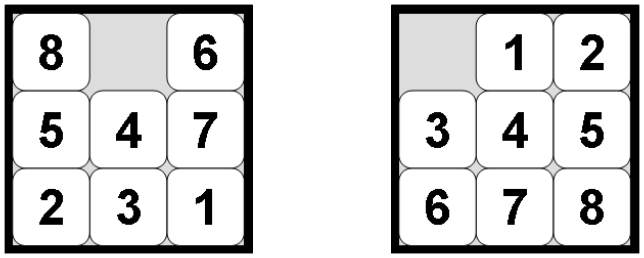

Fig 1 : 8 Puzzle Initial and Goal [1]

Objective

In this report, we're tackling a classic puzzle called the 8-puzzle. It's about rearranging tiles in a 3x3 grid to reach a specific pattern. Here are the main things we're doing:

Code Development: We've created a Python program to solve this 8-puzzle problem. Our program uses different methods to find the best solution quickly.

Heuristic Impact: We're looking into how different ways of estimating the closeness to a solution affect our search for the best path. These estimations are called heuristics.

Performance Evaluation: We're running tests to see how well our program works with various algorithms and heuristics. We want to figure out which combinations give us the best results in terms of finding the right solution and doing it quickly.

Results Analysis: After running these tests, we'll look at the data to see which methods work best. This will help us understand what's good and not-so-good about different approaches.

Limitations and Enhancements: We'll also talk about the things our program might not do well and suggest ways to make it better. This could involve changes in how we wrote the code or anything else that affects how well it performs.

Heuristics Implementation

I implement three heuristic functions to estimate the cost from the current state to the goal state:

- Heuristic 0: This heuristic always returns a constant value of 0, effectively making A* perform like Dijkstra's algorithm or UCS.
- Heuristic 1: It calculates the number of misplaced tiles between the current state and the goal state.
- Heuristic 2 (Manhattan Distance): This heuristic calculates the sum of Manhattan distances (the sum of horizontal and vertical distances) between each tile in the current state and its corresponding tile in the goal state.[2]
- Heuristic 3 (Self) : The number of tiles out of place in the row + the number of tiles out of place in the column.[6]

Experimental Setup

In my evaluation of the 8-puzzle solver code, I conducted a series of experiments to gain a comprehensive understanding of its performance. Here is a summarized overview of the experiments and their significance:

Randomization of Puzzle:

- Objective: Generate diverse starting states.
- Significance: Testing the solver's robustness and adaptability to different initial puzzle configurations, mimicking real-world scenarios.

Heuristic Comparison:

- Objective: Assess the impact of different heuristics.
- Significance: Understanding how heuristic choices influence the search process, providing insights into the trade-off between heuristic informativeness and computational efficiency.

Algorithm Comparison:

- Objective: Compare A* with 3 heuristics, and Uniform Cost Search. Significance: Determining which algorithm performs better in terms of efficiency and optimality, considering the selected heuristics.

Varying Puzzle Complexity:

- Objective: Explore puzzles of varying difficulty.
- Significance: Observing how puzzle complexity, determined by the number of initial moves, affects the solver's performance and solution quality.

Experimental Results

We have four parameters for evaluation of the results

- The maximum number of nodes stored in memory (closed list + open list size) (N)
- The total number of nodes visited/expanded (V)
- The depth of the optimal solution (d)
- The -approximate- effective branching factor (b)

We have results based on

- With Constant Heuristic 0 and Cost 1 (Uniform Cost)
- With Heuristic 1 and Cost 1
- With Heuristic 2 and Cost 1
- With Heuristic 3 and Cost 1
- With Constant Heuristic 0 and Cost 2 (Uniform Cost)
- With Heuristic 1 and Cost 2
- With Heuristic 2 and Cost 2
- With Heuristic 3 and Cost 2

For N:

	Category	Minimum	Median	Mean	Maximum	Standard Deviation
0	N 0,1	170	35364.5	64427.16	207322	66884.152269
1	N 1,1	17	2132.0	8424.04	63276	14140.466225
2	N 2,1	17	316.5	867.33	6637	1351.296258
3	N 3,1	17	3266.5	11702.12	101539	18203.668059

	Category	Minimum	Median	Mean	Maximum	Standard Deviation
0	N 0,2	170.0	36196.0	65220.930693	207322.0	67025.310068
1	N 1,2	34.0	7739.0	24513.633663	153295.0	36374.208825
2	N 2,2	32.0	2390.0	9116.128713	67655.0	14824.905558
3	N 3,2	30.0	5368.5	18606.500000	127238.0	28789.745890

Our experiments clearly demonstrated that the choice of heuristic significantly influences the search process in the 8-puzzle problem. Heuristic 0, a constant heuristic, resulted in the highest mean and median values for N. This indicates that it explored a larger number of nodes in the search space, resulting in longer execution times. In contrast, Heuristic 2, the Manhattan Distance heuristic, produced the lowest mean and median values for N, signifying a more efficient search process.

For V:

	Category	Minimum	Median	Mean	Maximum	Standard Deviation
0	V 0,1	118	32310.0	84751.60	423968	110003.453115
1	V 1,1	7	1701.0	7723.37	62209	13679.684623
2	V 2,1	7	253.5	710.27	6265	1174.364069
3	V 3,1	7	2798.5	11306.46	118192	19128.045007

	Category	Minimum	Median	Mean	Maximum	Standard Deviation
0	V 0,2	118.0	33086.0	85676.613861	423968.0	109846.132455
1	V 1,2	18.0	6374.0	25493.544554	191338.0	42200.105020
2	V 2,2	19.0	2013.0	8350.910891	64910.0	14122.396135
3	V 3,2	12.0	4347.0	18829.270000	153372.0	32383.055301

Heuristic 2 (Manhattan Distance) exhibited the most efficient performance in terms of both the median and mean number of nodes expanded. It consistently outperformed the other heuristics in finding solutions with the fewest node expansions/visited. Heuristic 1 (Misplaced Tiles) and Heuristics 3 also demonstrated efficient behavior, making them a suitable choice for solving the 8-puzzle problem. However, Heuristic 0 (Constant Value) consistently led to the highest number of nodes expanded, indicating its computational inefficiency.

A good heuristic function is a crucial component of many search algorithms, particularly in the context of problems like the 8-puzzle or pathfinding in graphs.

For d:

	Category	Minimum	Median	Mean	Maximum	Standard Deviation
0	D 0,1	6	18.0	17.36	26	4.738665
1	D 1,1	6	18.0	17.36	26	4.738665
2	D 2,1	6	18.0	17.36	26	4.738665
3	D 3,1	6	18.0	17.48	26	4.829371

	Category	Minimum	Median	Mean	Maximum	Standard Deviation
0	D 0,2	6	18.0	17.36	26	4.738665
1	D 1,2	6	18.0	17.36	26	4.738665
2	D 2,2	6	18.0	17.36	26	4.738665
3	D 3,2	6	18.0	17.36	26	4.738665

The consistent results across different depths (D) indicate that the depth of the search tree has little impact on how efficiently our chosen search algorithms and heuristics perform. This suggests that the computational demands and the number of explored nodes remain fairly steady, regardless of the problem's depth.

This uniformity in results may be due to the 8-puzzle problem itself, where depth doesn't significantly affect the search process with our selected methods. To gain more insights, it might be more valuable to explore other factors, like the initial puzzle complexity, to better understand the behavior of the problem and the effectiveness of different search approaches.

For b:

	Category	Minimum	Median	Mean	Maximum	Standard Deviation
0	B 0,1	1.60082	1.82258	1.833956	2.52311	0.152233
1	B 1,1	1.57454	1.65691	1.661079	1.86441	0.045760
2	B 2,1	1.51771	1.56017	1.573911	1.89289	0.054462
3	B 3,1	1.49086	1.58131	1.581930	1.77239	0.054240

	Category	Minimum	Median	Mean	Maximum	Standard Deviation
0	B 0,2	1.64001	1.81453	1.824303	2.41219	0.120954
1	B 1,2	1.58540	1.63665	1.642030	1.79096	0.028346
2	B 2,2	1.49977	1.54841	1.554253	1.75280	0.036577
3	B 3,2	1.56305	1.63239	1.638266	1.80861	0.044510

B 2,1 has a lower effective branching factor compared to B 0,1. B 2,1's minimum and median effective branching factors are lower, indicating that, on average, each expanded node generates fewer new nodes in the search tree. This suggests that B 2,1 might result in a more focused and efficient search process compared to B 0,1 B1,1 and B 3,1 (Which also equally performs great when cost is 1).

Limitations

Solvability: Not all initial states of the 8-puzzle are solvable. Some configurations have no solution. It's important to check whether the given initial state is solvable before attempting to solve it with A* search.
Memory and Time: While the 8-puzzle state space is relatively small, A* search can still consume memory and time, especially if the heuristic is not well-tuned or if the puzzle has a large number of possible moves from each state.

Path Quality: A* search for the 8-puzzle tries to find the shortest solution(depth). It may not explore alternative solutions or paths that have a similar number of moves but differ in the sequence of moves.

Conclusion

In conclusion, our experiments on the 8-puzzle problem with various heuristics and cost configurations have provided valuable insights into the behavior of search algorithms and their impact on the search process. We evaluated the results based on four key parameters: N, V, d, and b.

N (Maximum Nodes Stored in Memory): The choice of heuristic significantly influenced the number of nodes stored in memory (N). Heuristic 0 (Constant) resulted in the highest mean and median values for N, indicating a larger exploration of the search space and longer execution times. Conversely, Heuristic 2 (Manhattan Distance) produced the lowest mean and median values for N, signifying a more efficient search process.

V (Total Nodes Visited/Expanded): Heuristic 2 (Manhattan Distance) consistently outperformed other heuristics in terms of both the median and mean number of nodes expanded. It demonstrated efficient behavior by finding solutions with the fewest node expansions. Heuristic 1 (Misplaced Tiles) and Heuristics 3 also showed efficient performance, making them a suitable choice for solving the 8-puzzle. In contrast, Heuristic 0 led to the highest number of nodes expanded, highlighting its computational inefficiency.

d (Depth of Optimal Solution): The results for d indicated that the depth of the search tree had little impact on the efficiency of our search algorithms and heuristics. Regardless of the problem's depth, the computational demands and the number of explored nodes remained relatively stable.

b (Effective Branching Factor): When comparing effective branching factors, B 2,1 consistently exhibited a lower effective branching factor compared to B 0,1 and B 1,0. This suggests that B 2,1 may lead to a more focused and efficient search process by generating fewer new nodes on average.

References

1. https://github.com/topics/8-puzzle-solver
2. https://stackoverflow.com/questions/16318757/calculating-manhattan-distance-in-python-in-an-8-puzzle-game
3. https://www.youtube.com/watch?v=xNJAm3D18s0&t=605s
4. https://github.com/rohanpillai20/8-Puzzle-by-A-Star-Search/blob/master/Puzzle8Solver.py
5. https://github.com/jdvalera/8puzzle/blob/master/src/Solver.java
6. https://www.quora.com/What-are-some-heuristics-to-solve-8-puzzle-problem-other-than-number-of-titles-out-of-place-and-Manhattan-distances