CS170: Artificial Intelligence

Project 01: Eight Puzzle

Prof. Niloofar Montazeri

Otniel Thehumury	Matthew Walsh
otheh001@ucr.edu	mwals003@ucr.edu
862029595	862088280
05/02/2021	05/02/2021

- -

Works Referenced:

UCR CS170 lecture slides were referenced while coding the three searches:

- Slides 02 (Blind Search) were referenced in implementing the Uniform Cost Search.
- Slides 03 (Heuristic Search) were referenced for the Misplaced tile heuristic.

All major code is original. The following references were used in non-major subroutines:

- Referenced Python PEP 209 for Python functions involving arrays: https://www.python.org/dev/peps/pep-0209/
- Referenced delftstack.com on how to clear the console in Python: https://www.delftstack.com/howto/python/python-clear-console/
- Referenced w3schools.com for acquiring user input in Python: https://www.w3schools.com/python/python_user_input.asp

These references were only used for general Python methods. No references outside of course materials were used in implementing search algorithms and solving the Eight-Puzzle problem.

The code to reverse the parent nodes to display the correct path is based on Otniel Thehumury's submission to LeetCode for a similar problem. This resource was only used as reference for this non-major subroutine, and the code we implemented is original.

Version History:

0.1 - 04/26/2021	Initial separate versions from each partner.
0.2 - 04/27/2021	Merged versions with code from both partners.
0.3 - 04/28/2021	Joint version is functional, but with test output.
0.4 - 04/29/2021	Misplaced tile heuristic is fully implemented.
0.5 - 04/30/2021	Trace completed but not fully correct.
0.6 - 05/01/2021	Node expansion fully implemented.
1.0 - 05/01/2021	Completed version with correct trace output.
1.1 - 05/02/2021	Final completed version, ready to submit.
1.2 - 05/02/2021	README, trace outputs, and report completed.

CS170 Project 01 Report

Otniel Thehumury / SID: 862029595
Matthew Walsh / SID: 862088280

Overview:

This project investigates the performance of the three search algorithms on a simple eight-puzzle problem. We explored the efficiency of a uniform cost search, an A* search with a misplaced tile heuristic, and an A* search with a Euclidean distance heuristic.

Design:

In designing our program, we chose to implement a graph search algorithm. We agreed that keeping track of explored nodes would allow a much more simple and efficient implementation than a tree search. While graphs, like eight-puzzles, are prone to loops, keeping track of explored states helped us prevent this setback.

We approached the uniform cost search with a simple breadth-first search in mind. It was not until we added state costs that our function evolved into a fully-fledged uniform cost search.

We initially implemented a search by Manhattan Distance before switching to the Misplaced Tile Heuristic. Our Manhattan distance heuristic performed similarly to the Euclidean distance heuristic, but ultimately did not meet project requirements. Changing to the misplaced tile heuristic was a simple implementation. We placed the heuristic in its own function to take advantage of an easily-modifiable goal state.

In displaying the nodes, we opted for a more complex yet rich display. Categorizing and dividing sections of our output allowed the data to be more digestible. We opted to provide the trace of all nodes explored, as well as a trace of the correct path at the algorithm's conclusion.

Algorithms:

The uniform cost function was the simplest to implement, due to the cost function remaining the same for each node explored. Despite its completeness, this algorithm proved to be the least efficient. While this algorithm completed the eight-puzzle in a timely manner, it may prove to be useless with more complex problems.

Implementing the A* search with the misplaced tile heuristic was more challenging to implement, but provided a much more efficient search solution. This heuristic would be much more efficient than the uniform cost search in larger problems. Implementing the euclidean distance heuristic required a simple modification to our A* search algorithm. Instead of appending the number of misplaced tiles to our heuristic measure, we calculated the Euclidean distance between a tile's current and goal states. This heuristic proved to be the most efficient.

Observations:

The misplaced tile heuristic and the Euclidean distance heuristic proved to be similarly efficient. For both heuristics, our test cases produced similar values for nodes explored and maximum queue size. It was not until the problem space became exponentially greater (see “Oh Boy” case) that Euclidean distance proved to be more efficient.

On the other hand, the uniform cost search proved to be enormously inefficient by comparison. Our simpler test cases took uniform cost search three to four times longer to find the result than our two A* searches. The “Oh Boy” case took an incredibly long time to execute, and the impossible problem remained unsolvable by our own algorithm. When attempting to solve the impossible problem, Microsoft Visual Studio killed our program after some time.

The following charts display the results of our algorithm on the provided eight-puzzles:

Nodes Expanded

	Uniform Cost Search	Misplaced Tile Heuristic	Euclidean Dist. Heuristic
Trivial	0	0	0
Very Easy	1	1	1
Easy	6	2	2
Doable	16	4	4
“Oh Boy”	94088	9103	1058
Impossible	∞ (Infinite)	∞ (Infinite)	∞ (Infinite)

Max Queue Size

	Uniform Cost search	Misplaced Tile Heuristic	Euclidean Dist. Heuristic
Trivial	1	1	1
Very Easy	3	3	3
Easy	8	3	3
Doable	18	4	4
Oh boy	24983	5051	591
Impossible	∞ (Infinite)	∞ (Infinite)	∞ (Infinite)

Challenges & Future Work:

Tracking nodes and testing our functions was difficult at first. Keeping track of explored nodes and the result of heuristic calculations relied on excessive printed output. Collaborating remotely with a partner on delicate and precise graph traversals provided an additional challenge.

Given the opportunity, there are a few additional improvements we could add to our functionality. For example, we could provide users with the option to perform a tree search instead of a graph search. We could also provide the user with the opportunity to compare the efficiency of each algorithm on a single puzzle.

Conclusion:

This project was helpful in displaying the differences between the various search algorithms used in artificial intelligence. Each algorithm has strengths and weaknesses, and each is strengthened or limited by the type of problem they are used to solve for. Crucial AI concepts were learned through a simple yet realistic application of search algorithms.

Notes to Grader:

One of our test machines had difficulty producing the full printed output. While the backend functionality performs as expected, it is possible that some Python configurations will have difficulty replicating our trace.

Additionally, the extra credit output has been commented out within our code. We placed our two traces within PDFs, rather than text files, for uniformity of filetype and ease of display.