CS170 8-Puzzle Solver

https://github.com/vinayakgajjewar/CS170-8-Puzzle
This project uses Python 3.10.6. To run the solver, execute the following command and follow the prompts.

```
$ python3 main.py
```

Design

For this project, I implemented graph search instead of tree search. I implemented the explored set using a native Python list and the frontier as a priority queue using the `heapq` package.

Challenges

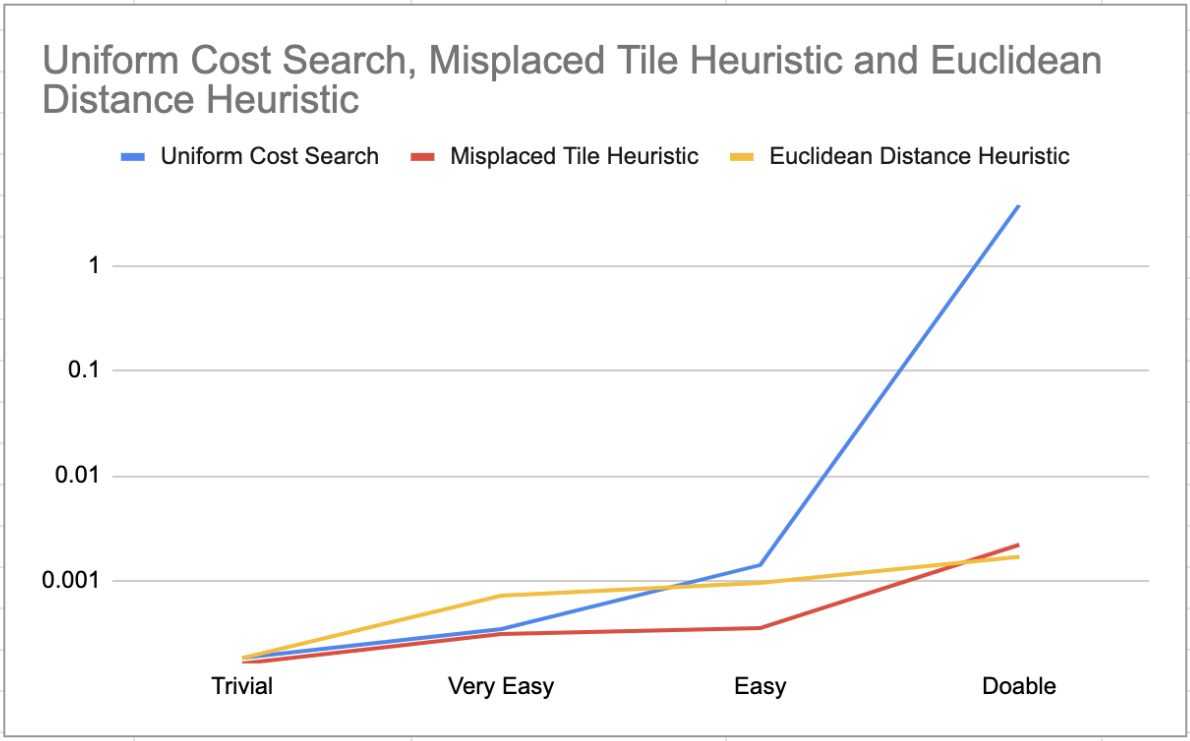
One challenge I faced while coding this project is that early on in the project, I hard-coded the number of rows of the puzzle to equal three. This decision made the code much longer and much less abstract than it could have been. Another challenge I faced was figuring out when Python is pass-by-reference or pass-by-value. Many bugs I faced while coding this project came from mistakenly modifying a value across variables instead of performing a shallow copy.

Heuristic function comparison

While testing the solver, I found that the Euclidean distance heuristic outperforms the misplaced tile heuristic on both the number of expanded nodes and the maximum frontier size. I also found that the heuristic choice only matters for the more complex initial configurations. Any heuristic will yield approximately the same result if the problem is reasonably trivial.

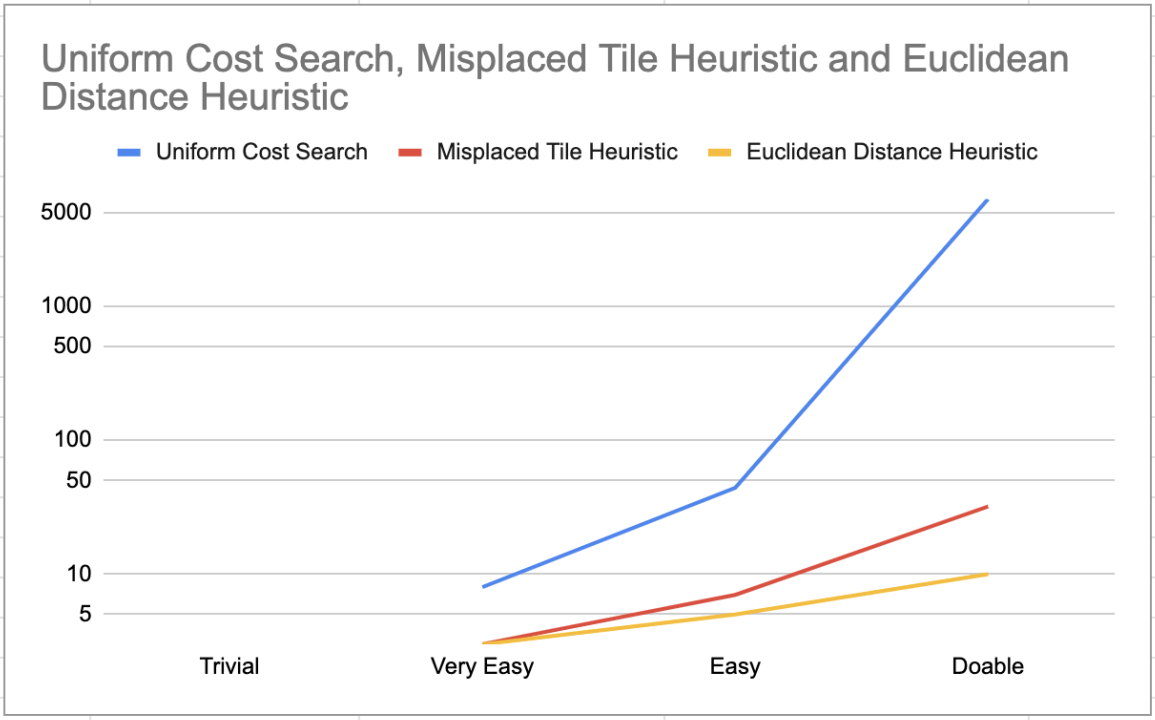
Findings

	Uniform Cost Search	Misplaced Tile Heuristic	Euclidean Distance Heuristic
Trivial	0.000185251236	0.0001640319824	0.0001840591431
Very Easy	0.0003471374512	0.0003130435944	0.0007228851318
Easy	0.00141620636	0.0003559589386	0.0009570121765
Doable	3.785580158	0.002205848694	0.001698970795
Oh Boy			
Impossible			



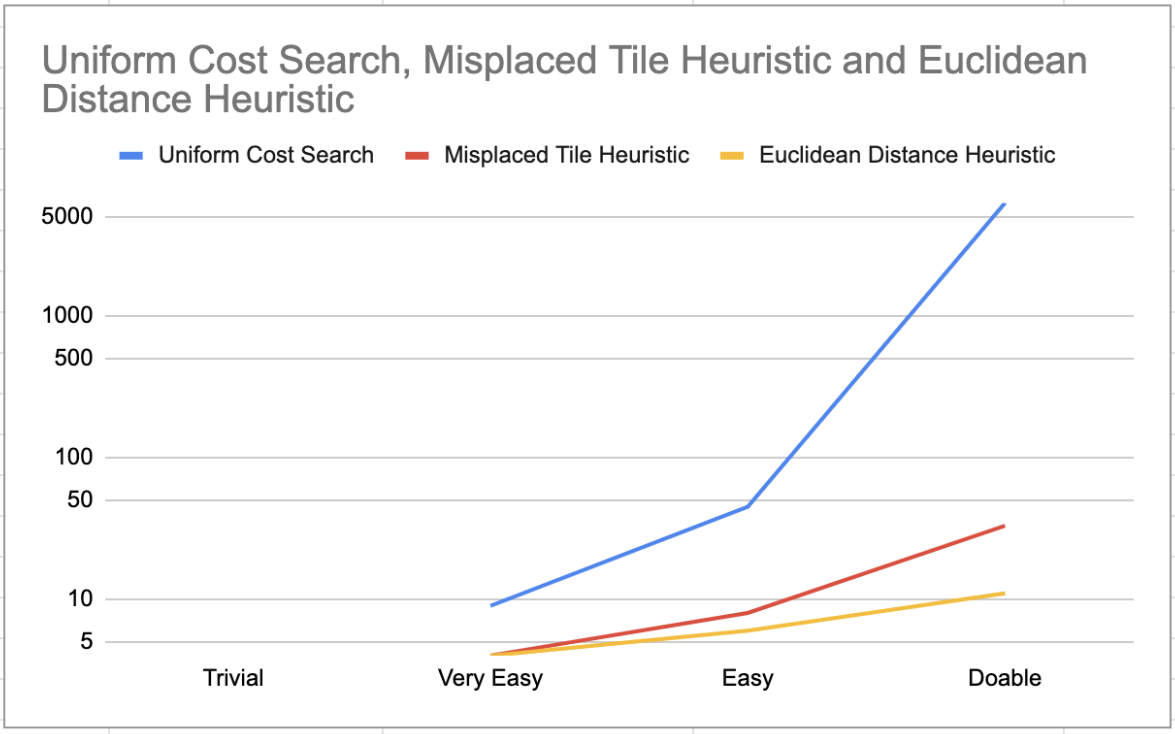
The above graph displays the relationship between heuristic choice and execution time for different puzzle difficulties. The above chart demonstrates how the choice of heuristic matters less for easy puzzle configurations. However, as the puzzle difficulty increases, the choice of heuristic becomes more critical. Uniform cost search is the least efficient as we get to more difficult puzzles. The Euclidean distance and misplaced tile heuristics have approximately the same time complexity, but the Euclidean distance heuristic is slightly better.

	Uniform Cost Search	Misplaced Tile Heuristic	Euclidean Distance Heuristic
Trivial	0	0	0
Very Easy	8	3	3
Easy	44	7	5
Doable	6288	32	10
Oh Boy			
Impossible			



The above graph displays the relationship between heuristic choice and the number of expanded nodes for different puzzle difficulties. The most efficient heuristic is the Euclidean distance heuristic, followed by the misplaced tile heuristic. Uniform cost search is the least efficient in this regard. The misplaced tile heuristic and the Euclidean distance heuristic are roughly equivalent to easy puzzles. Still, as the puzzles get more complex, the choice of heuristic matters more.

	Uniform Cost Search	Misplaced Tile Heuristic	Euclidean Distance Heuristic
Trivial	0	0	0
Very Easy	9	4	4
Easy	45	8	6
Doable	6289	33	11
Oh Boy			
Impossible			



The above graph displays the relationship between heuristic choice and maximum frontier size for different puzzle difficulties. The most efficient heuristic is the Euclidean distance heuristic, followed by the misplaced tile heuristic. Uniform cost search is the least efficient in this regard. The misplaced tile heuristic and the Euclidean distance heuristic are roughly equivalent to easy puzzles. Still, as the puzzles get more complex, the choice of heuristic matters more.

Trace

```
(base) Vinayaks-MacBook-Pro:CS170-8-Puzzle vinayakgajjewar$ python3 main.py
Welcome to vgajj002's 8-puzzle solver.
Type '1' to use a default puzzle or type '2' to enter your own puzzle.
2
Enter your puzzle, using a 0 to represent the blank.
Enter the first row, using spaces between numbers: 1 0 3
Enter the second row, using spaces between numbers: 4 2 6
Enter the third row, using spaces between numbers: 7 5 8
Enter your choice of algorithm.
1. Uniform cost search (h(n) = 0).
2. A* with misplaced tile heuristic.
3. A* with Euclidean distance heuristic.
3
Popped the following node from the frontier (g(n)=0, h(n)=5.23606797749979):
1 0 3
4 2 6
7 5 8
Popped the following node from the frontier (g(n)=1, h(n)=3.414213562373095):
1 2 3
4 0 6
7 5 8
Popped the following node from the frontier (g(n)=2, h(n)=2.0):
1 2 3
4 5 6
7 0 8
Popped the following node from the frontier (g(n)=3, h(n)=0.0):
1 2 3
4 5 6
7 8 0
We have found the solution.
The solution is at depth 3.
Solution path:
Start -> blank down -> blank down -> blank right
We expanded 10 nodes.
The max number of nodes in the frontier at any one time was 11.
Time taken: 0.0017387866973876953.
(base) Vinayaks-MacBook-Pro:CS170-8-Puzzle vinayakgajjewar$
```