CMPT-225: Final Project

Spring 2023

Nicholas Mah

301427489

Algorithm

My solver utilizes a Greedy/Pure-Heuristic Search for all puzzle sizes with no segregation of the board or solving the top row and column to reduce the board to a smaller size. I had initially tried IDA* and A* which yielded more optimal solutions, but after I could no longer gain any more significant speed improvements, I switched to PHS. A* with its tracking of depth meant it would be able to obtain a shorter path to the goal state, but this hindered its overall speed. IDA* I did not spend as much time experimenting with and my heuristic at the time was not as substantial. One of the advantages of IDA* was its memory performance, however, with my current solution with PHS, I have no memory issues for all sizes of puzzles.

During the beginning with A* I had experimented with solving the top row and column to reduce the board size for the larger instances of puzzles. It created longer solutions, but for these larger instances of the board there was lots of wasted moves being checked in the “dead space” of the bottom right square. If the move applied didn’t affect any of the tiles in the upper edge of the puzzle, then the heuristic value wouldn’t change. Scaling this issue up to the larger puzzles resulted in long performance times and memory issues with how many nodes were being added to the open set. I tried adjusting my heuristics to add weight to these moves in the “dead space” but found no success.

I stuck with PHS as it was the fastest with my heuristics and from there, I further optimized my program and improved my heuristics.

Closed Set / Hash Function

I used a HashSet to track my closed list of nodes. This contained the hashcodes of the puzzle nodes. As I was only holding one value for each object a HashSet was suitable and provides fast processing. I had initially used a HashMap at the beginning when I was using A* as I needed to track depth values alongside the node.

The hashing function provided significant gains in performance for me after using a purely multiplicative approach. I took inspiration from the Zobrist Hash algorithm and if the board size is large enough, I will apply a scaling factor which eliminated collisions.

Heuristics

The current program utilizes the following heuristics:

- Manhattan Distance
- Hamming Distance
- Euclidean Distance
- Linear Conflicts

I initially started with Manhattan Distance and that was sufficient up to 4x4. Using a single heuristic, Manhattan was the only one that was able to quickly solve a 4x4 puzzle and that became the backbone for my heuristic function. I added Hamming Distance and Linear Conflicts which allowed me to solve 5x5's instantly. After experimenting and playing around with the puzzle myself, I started to change the weight of the 3 heuristics. This allowed me to solve all puzzles. The very large boards were still a bit slow, but adding a weighted Euclidean Distance significantly sped up my solver. In combination, these 4 heuristics with different weights provided me the greatest performance.

A significant optimization I was able to implement was combining the calculation of Manhattan, Hamming and Euclidean Distance together. All 3 loop through the board and Manhattan/Euclidean both use the same values for their calculations.

Performance

I'm able to solve up to 6x6 instantly. On average the rest of the puzzles are solved in a couple seconds. 2 of the 9x9 boards; board37 and board38 take a little more than 30 seconds on csil.

My Journey

I was able to implement my Puzzle class easily to provide me all the functions I required. I had made a list of all the functions I required and made functions for solving top row/column which after testing I no longer needed and deleted.

The algorithm was also quick to implement for me and early on I had coded A*, PHS and IDA*. Having all 3 available to me allowed me to test and experiment and decide on which to commit to. This ended up being PHS.

The part that took the most time was the heuristics; Experimenting with which heuristics to use, what combinations, and eventually what weights should be applied.