ecm2423 - A*

March 2022

1 8-puzzle as a Search Problem

In computation theory search problem is a task of finding a specific structure, element, in a larger structure of structures. It is possible to view 8-puzzle problem as a problem of finding such ordered set, *path*, of configuration transformations, *moves*, that leads from a specific configuration of the 8-puzzle board, *picture*, to a different picture.

2 Solving 8-puzzle using A*

2.1 What is A*?

The A* algorithm is a search algorithm. It can be used to traverse the tree created by applying the four possible moves to the current picture in an efficient manner.

It is an alternative to applying all possible moves to the picture, checking the resulting pictures, and if none leads to a satisfying result, applying all applicable moves to new pictures and checking those until a satisfying result is produced. (Width-first algorithm.)

Or going completely down a specific route until a cut-off point or a satisfactory picture is reached. (Depth-first algorithm.)

Rather, A* uses a sum of *heuristic*, more on that later, and distance, number of moves, between the original picture and current picture, to determine which node to expand next. The children are evaluated through heuristic and added to the queue of nodes to be expended.

2.2 Heuristic

A heuristic is a mathematical formula that estimates how many moves are necessary to before the desired picture can be reached. Its purpose is to prevent expanding nodes that are further away from the desired picture than the current best option.

2.2.1 Admissible heuristic

Heuristic has the admissible property. A heuristic is admissible iff it underestimates the number of moves necessary to get the desired picture. If a heuristic is admissible, it guarantees that the final path is optimal.

2.3 Heuristics used

There are five Heuristics available in the program.

1. Trivial Heuristic - $f() = 0$. Trivially admissible heuristic that effectively, (not actually [1]) turns the A* search algorithm into a breadth-first algorithm.

2. Euclidean distance for the empty tile - This heuristic estimates the distance of the picture from the desired picture as a distance between the empty tile's current location and its location in the desired picture. This Heuristic is admissible since the tile needs to be in the location it has in the desired picture. In order to move there, $N > Euclidean\_distance$ moves need to be performed. There is no special quality in the empty tile in regards to this heuristic.

3. Manhattan distance for the empty tile - This heuristic stems from the same idea as Euclidean distance for the empty tile, but instead Manhattan distance, sum of the distances along each axis, is used. This heuristic should be slightly more optimal, as $M_0() >= E_0()$

4. Euclidean distance for all tiles - Further expansion on the second heuristic. Instead of being limited to a single tile, now all tiles are evaluated and their distance from their location in the desired picture is summed. This heuristic is still admissible, as each of these tiles need to be moved to their location in the desired picture, and the number of moves is lower or equal to their distance.

5. Manhattan distance for all tiles - Combination of the third and fourth idea. Now the distance evaluation is expanded to all tiles and Manhattan distance is used as a more accurate measure than the Euclidean one.

3 Initial state

3	5	1
6	4	0
7	8	2

Table 1: Initial state

Initial state used for evaluation of the program characteristics can be seen in table 1. The picture has 11 moves complexity.

4 Code

Note that the code is not included in this report. The code files are attached in the .zip file in their own folder as the codebase consists of 700 lines across multiple files.

5 Performance Analysis

5.1 Case for custom Initial state

The custom 11 moves picture was chosen as a memory required to store large trees of less efficient heuristics gets exorbitantly large, with memory required to store a large tree taking upwards from 40GB. This is an optimisation issue beyond the scope of the assessment.

5.1.1 Possible optimisation

Possible ways to mitigate that would be reducing GameState and GameStateNode classes into a single one, or (if computation power is more abundant than memory) having nodes store only a transformation performed on their parent, and create the game state ad-hoc.

Another possible optimisation would be deleting children that reached the maximum depth without succeeding and deleting their parents, (and their parents recursively,) once orphaned. This approach would be more efficient on an algorithm that favours depth when breaking ties.

5.2 Observed Performance

Because the program runs in a multi-threaded mode, which is non-deterministic by design, and on a multi-tasking machine, five runs were performed and recorded, (their .log files are included in the .zip file,) and their performance averaged to account for disruptions caused by processor scheduling.

Run	Trivial Heuristic	Euc - Empty	Mnh - Empty	Euc - All	Mnh - All
1	1.0685813	1.2492986	1.0957985	0.1733666	0.3867348
2	1.094301	1.5532791	1.3595367	0.2357036	0.5052109
3	1.0527986	1.2698566	1.0870824	0.1714249	0.3937497
4	1.103057	1.2783032	1.1153026	0.1598163	0.3797943
5	1.0677075	1.3716715	1.1351444	0.1391878	0.3928824
Average	1.077	1.344	1.159	0.176	0.412

Table 2: Run times in seconds on custom initial state

Run	Mnh - All	Euc - All
1	3m39s	∞
2	3m36s	∞
3	3m45s	∞
Average	3m40s	∞

Table 3: Run times in seconds on default initial state

Note that only the four most significant digits were included for the average. Two interesting trends can be observed in the table 2:

a) The benchmark trivially admissible heuristic performed better than the two heuristics reliant only on a single tile.

b) Sum of Euclidean distances for all tiles performed better than the sum of Manhattan distances for all tiles. It finished in less than half the time.

Item a) is interesting, one possible explanation is that the additional overhead caused by evaluating the heuristic outweighed the gain in efficiency from using a heuristic in the first place. In light of that, a possible explanation for b) is that calculating the absolute value is less computationally efficient than calculating a square root of two (non-identical) squares, which seems counter-intuitive. b) warrants further investigation.

6 Note on General Solution

Not all pictures are solvable. If the number of inversions, (numbers changing order in the desired picture along any axis) is odd, the picture does not have a solution. The program does not employ any checks to prevent executing on unsolvable puzzle to allow for extreme testing by deploying the code on unsolvable picture.

References

[1] Holte, R. C. Common misconceptions concerning heuristic search. *Proceedings of the Third Annual Symposium on Combinatorial Search (SOCS-10)* (8 2010), 46–51.