The University of Melbourne

COMP30024 Artificial Intelligence,

Project Part A.

Authors: Isobel Byars, Melvin Hartley

Unit Coordinator: Professor Chris Leckie

April 5, 2022

1. **How did you implement the A* search? Discuss implementation details, including choice of relevant data structures, and analyse the time/space complexity of your solution.**

The A* search was implemented by representing each node to be expanded in the following data structure:

[[i,j], [m, n], g, f]

Where (i,j) are the coordinates of the current cell, (m,n) are the coordinates of the preceding cell along the shortest discovered path to (i,j), g is the actual cost of reaching (i,j) from the start cell, and f is equal to g plus the estimated cost of reaching the goal from (i,j) (using the heuristic discussed in question 2). The program maintains two lists of nodes: a list of unexplored nodes, and a list of explored nodes. The program explores unexplored nodes in ascending order of f_value, meaning that an optimal solution will always be found.

* Space complexity: each node explored will generate a maximum of 6 more nodes in memory, and will not be erased until the program terminates. However, there will never exist more than one node per square on the board, meaning space complexity is O(n^2) for an n x n board.

* Time complexity: In the general A* search problem, the time complexity is exponential in (e.d), where d is the cost of the shortest solution and e is the relative error in the heuristic function.

The time complexity of our code can be summarised as follows:

`while unexplored:`	In the worst case all n^2 nodes will be explored; $O(n^2)$ complexity
`unexplored = sorted(unexplored, key=itemgetter(3))` `current = unexplored.pop(0) # [[4, 2], [0, 0], 0,`	Employing python's sorted() algorithm introduces a complexity of $O(n^2 \log n^2)$ [(1)] which in effect is $O(n^2 \log n)$
`for k in range(6):`	Time complexity of the remaining steps is trumped by preceding ones, thus are not considered in the calculation

Finally, in the worst case, the sort occurs once for each node in the board space bringing out full worst-case complexity to

$$O(n^2) * O(n^2 \log n) = O(n^4 \log n)$$

This complexity is still polynomial and not exponential implying that our A* algorithm is more time-efficient than the general case.

[1] https://stackoverflow.com/questions/14434490/what-is-the-complexity-of-the-sorted-function

2. What heuristic did you use and why? Show that it is admissible, and discuss the cost of computing the heuristic function, particularly in relation to the overall cost of the search.

The heuristic we have chosen is a generalised Manhattan-like distance specific to the hexagonal coordinate system. It takes the distance to be the maximum values out of (|i-m|, |j-n|, | (i-m) + (j-n)|)

* The heuristic is admissible because the heuristic will never overestimate the cost of reaching the goal. It represents the minimum distance possible on an empty grid, and adding obstacles will only increase the true path cost.

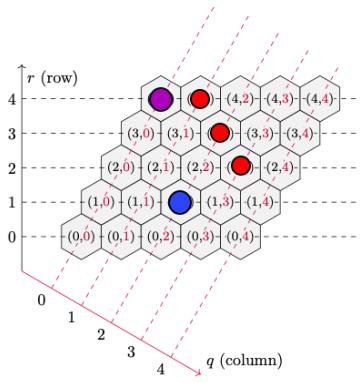
* This measure is extremely cheap to calculate, especially in comparison to the A* search. The time complexity is constant.

* This heuristic is both admissible and consistent, so nodes will not need to be reopened to find an optimal solution. This further reduces the overall cost of the A* search (However, in our solution we are still checking nodes to be reopened in case we decide to change the heuristic in part B of the project).

3. (Challenge.)

There would be two components to an alteration of our solution. The first would be to assign all newly discovered clusters of occupied nodes with the path cost of the cheapest parent. For each of these occupied nodes, you would still consider the application of a heuristic function and sort them accordingly.

An admissible heuristic function that could be used for this situation would be to take the Manhattan distance for each tile and subtract the number of 'unexplored' occupied tiles from this value. In this manner, the heuristic would no longer overestimate the path cost, as in the best case scenario all the unaccounted for unexplored occupied tiles could form a chain directly from the node in question towards the goal tile, reducing the theoretical Manhattan distance by the number of those tiles. Finally, if this number is negative due to situations with a large number of 'unexplored' occupied tiles, the heuristic should simply return zero.



In this case, however, our previous Manhattan heuristic would no longer be admissible. Take the following example, when calculating the cost of the cheapest path from the blue cell to the purple cell, our heuristic estimates the cost to be three. However, as evident in the example, this overestimates the true cost of one using the existing pieces.