CS 404 HW1

Part 1: Modeling the Color Maze as a Search Problem:

States:

In my search problem model, the input and the states are a rectangular game board with a single agent. The maze and the states are defined as a 2D matrix, structured as lists inside a list in Python. Inside the lists, “0” denotes the cells that are empty, “X” denotes the cells occupied by walls, “S” denotes the cells that are colored, and “C” denotes the current position of the agent. Every state in our problem is defined as such a matrix.

Successor States:

In our maze problem, the agent can move in four cardinal directions: up, down, right, or left. Once a direction is chosen, the agent moves in that direction until it reaches a wall and colors all the cells it travels over. Once a cell is colored, its color does not change.

Successor State Function:

So, our success state function will be that, from where the “C” cell is located in our matrix, there will be 4 cardinal directions, as mentioned, that can be chosen. Among our cardinal directions, the directions that are not directing to the neighboring cell of the “C” cell that are denoted by "X,” which represent the walls, can result in a successor state. That is, directions that do not have a wall within a one-cell range of the agent can result in a successful state. When a valid cardinal direction is chosen, in the resulting successor state of our current state, all of the cells that are between the agent, which is the “C” cell, and the closest wall, which is an “X” cell that is in the direction of the chosen cardinal direction, will be denoted with "S,” including the “C” cell, except the cell that is the neighbor of the “X” cell that was in the chosen direction, which will be the new current position of the agent, hence will be denoted by “C”.

With this function, we can define every succeeding state given the current state for all the states in our problem.

Goal Test:

Our goal is to color every cell in the maze. So, with the definitions we made, the goal state is any state that has no cell denoted with "0,” which represents the empty cells, or a state where all cells are denoted with an "X,", "S,” or “C.”. When this condition is satisfied, that means the goal state is reached.

Initial State:

The initial state can be any given maze that satisfies the state definitions made and that should not have any cell denoted with “0” and must have a single cell denoted with “C.”.

Step cost function:

Since for our problem we want to solve the maze with the minimum amount of total distance traveled by the agent, the number of cells that the agent will travel upon reaching the success state will be our step cost function. That is, the distance in cells between the “C” cell in the current state and the “C” cell in the successor state is the step cost for every successor state.

Part 2: Defining the Heuristic Functions:

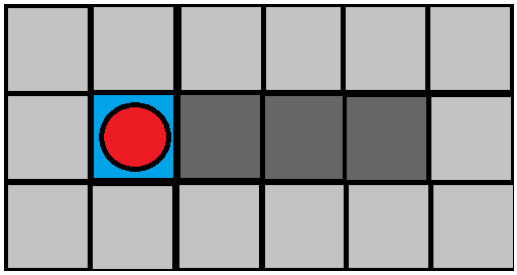
b) An admissible heuristic function for our problem is the number of remaining unpainted cells (“0”) in the succeeding state. This heuristic function is admissible and therefore never overestimates the cost to reach the goal state from any node. We can prove this function with proof by contradiction.

For the heuristic to overestimate ($h(n) > h*(n)$), it would imply that the agent can color more tiles in one move than there are moves available, which contradicts the established movement rule. Essentially, the agent cannot color more tiles than the paths it travels since each path corresponds to a single move, and each move colors a set of contiguous tiles. Each move can only color one straight path of tiles until a wall is encountered.

Since the agent cannot color more tiles than it can travel, the cost to reach the goal state is not overestimated, and thus the heuristic function is admissible.

a) I have first explained the admissible heuristic (h2) function because my inadmissible heuristic function is a weighted heuristic function of h1. I have decided to use this heuristic function in my implementation to observe the results of the memory and time consumption later on in my implementation. h1(n) is weighted by 1.5.

We know that h1 is inadmissible since it is a weighted heuristic function and thus overestimates the cost to reach the goal state from any node. In the maze illustration below, $h^*(n) = 3$ (since three moves are exactly needed to color all tiles). But in our h1 function, $h1(n) = 1.5 * h2(n) = 1.5 * 3 = 4.5$. Therefore, in this case, $h1(n) > h^*(n)$, which violates the condition for admissibility.



c) A heuristic h2 is monotone if, for every node n and every successor n' of n generated by any action a, the estimated cost of reaching the goal from n through n' is less than or equal to the cost of getting to n' from n plus the estimated cost from n' to the goal. This can be expressed as: $h2(n) \leq c(n, a, n') + h2(n')$.

In our h2 function, moving from any tile to an adjacent tile always incurs a fixed cost, and after any move, the heuristic value (number of uncolored tiles) can only decrease or stay the same. For any tile n and its successor n', the heuristic value at n is always greater than or equal to the heuristic value at n' plus the cost of the move. So, we can say that h2 is monotone.

Part 3: Implementing the A* search algorithm:

The Python code implementation and the solutions are given in the other zip document.

Part 4: Experimental evaluation of the A* search implementation:

b) Here is the table that shows the result of my experiments:

Maze Index	Size	Difficulty	Distance h2	Distance h1	Expanded Nodes h2	Expanded Nodes h1
0	144	Difficult	102	102	1414309	283358
1	144	Difficult	67	67	32131	1541
2	144	Normal	47	52	2025	291
3	144	Easy	24	24	53	18
4	144	Easy	28	28	42	20
5	144	Normal	61	61	39123	11185
6	144	Normal	61	66	8181	1841
7	144	Normal	74	74	21646	2953
8	144	Normal	55	55	49	19
9	144	Difficult	120	120	1639881	465761
10	144	Difficult	86	86	91215	19737
11	144	Difficult	77	86	100182	9770
12	144	Easy	22	22	17	6
13	144	Easy	24	24	48	10
14	144	Easy	45	48	480	20
15	144	Normal	74	81	1833	545

Maze Index	Size	Difficulty	CPU Time h2 (s)	CPU Time h1 (s)	Memory h2 (MB)	Memory h1 (MB)
0	144	Difficult	99.543010	27.687116	2114.269531	239.640625
1	144	Difficult	1.647087	0.125386	0.265625	0.007812
2	144	Normal	0.095179	0.017663	0.000000	0.003906
3	144	Easy	0.014016	0.000000	0.000000	0.000000
4	144	Easy	0.000000	0.000000	0.000000	0.000000
5	144	Normal	1.900010	0.814865	0.003906	0.015625
6	144	Normal	0.402347	0.133910	0.003906	0.000000
7	144	Normal	1.035356	0.202380	0.527344	0.011719
8	144	Normal	0.015652	0.000000	0.000000	0.000000
9	144	Difficult	108.356033	45.213683	1471.031250	60.980469
10	144	Difficult	4.857805	1.458168	0.000000	0.230469
11	144	Difficult	6.646519	0.871979	0.011719	0.015625
12	144	Easy	0.000000	0.000000	0.000000	0.000000
13	144	Easy	0.000000	0.000000	0.000000	0.000000
14	144	Easy	0.031253	0.000000	0.000000	0.000000
15	144	Normal	0.076123	0.031959	0.003906	0.000000

c) -What do you observe about the scalability of the A* search algorithm? How does the time and memory consumption increase as the input size increases? Are these observations surprising or expected, considering the asymptotic time and space complexity of the algorithm? Please explain.

The time and memory consumption appear to increase with the complexity and size of the mazes. This is expected as A* search's time and space complexity are both O(b^d) where b is the branching factor (the average number of successors per state) and d is the depth of the shortest path. Larger and more complex mazes will result in higher branching factors and deeper search paths, which leads to increased time and memory consumption.

Considering the scalability, we can see a clear difference between the two heuristic functions. The weighted heuristic shows overall better results considering time and memory consumption. This is expected since h1 will search the space more effectively, despite its lack of admissibility.

Scalability is not uniform; it can be significantly affected by the choice of our heuristic function. But overall, we can observe that both heuristics scale as expected with maze complexity. h1 will scale better in terms of time and memory consumption.

These observations are not surprising, as they align with the concepts of the A* search. h1 scaling better in terms of time and memory consumption is also expected since we know that the weighted heuristics are more optimal for exploring space while not guaranteeing the shortest path.

-How does the A* search algorithm explore the search space, with h1 vs. with h2? Are these observations surprising or expected, considering the admissibility/monotonicity features of the functions? Please explain.

With h1, the algorithm tends to explore fewer nodes compared to h2 especially in larger, more complex mazes. The lower node expansion of h1 results in a more efficient time and memory usage at the cost of accuracy. The h2 function does not overestimate the costs to reach the goal and thus provides a more thorough exploration. This exploration ensures the optimality of the found path.

These observations are again, expected based on the properties of the A* search and heuristics. Admissible heuristics, like h2 ensure that A* is both complete and optimal. However, they may explore a large search space. Non-admissible heuristics like h2 can be more optimized to explore fewer nodes as we can with weighted heuristics, but will not guarantee the optimal solution.

Our overall results show the trade-off between optimality and efficiency, as can be seen in many search algorithms, including the A* search.