CS404-Assignment 1
Solving Color-Maze Puzzle using A* Search

Görkem Yar 27970, Kaan Bilgili 27983

March 2024

1 Modeling the puzzle as a search problem

Color-Maze problem is a grid game with a single agent. The agent is capable of moving in the directions up, down, right, and left. Once the agent chooses a direction to move, the agent has to move until it reaches a border or a wall. The agent colors the cells that it has moved by. The goal of the agent is to color all the cells with minimum cost. The input of the program is a game board in the format of a grid where the cells are marked with X, S, or 0. S stands for the position of the agent (there can only be one S). X shows the walls (walls cannot be passed by the agent or be colored). 0 denotes for the uncolored cells. 1 represents the colored cells (Initially, there are no cells with 1).

1.1 States

The state of the search algorithm is equivalent to the state of the board (Grid). Different states can be accomplished by the different positions of the agent (the S symbol), the positions of the walls, and the cells that are colored and uncolored. The move of the agent changes the state of the program. To calculate state count:

- $N = row * col$
- B =Number of walls in the grid
- $(N - B) * 2^{N-B-1}$ The first N-B stands for the possible position for the agent. Later one denotes the possible values (0 or 1) for the remaining cells.

This formulation is not 100% accurate. All possible states cannot be achieved since the agent moves in a direction until it reaches a border or a wall. So intermediate states are not possible. However, this formulation will give a big-O estimation for the possible number of states. The number of states is in $O((N-B)*2^{N-B-1})$.

1.2 Successor State

There are 4 possible successor states for each state. These successor states can determined by the last action of the agent in the parent state. As already mentioned, the agent can take action in four cardinal directions: up, down, right, and left. With the action agent will move until it reaches a border or wall while coloring the cells. When it reaches a border or a wall the successor state is generated. However, it is not guaranteed that each action can create a successor. If the agent is next to a wall or a border, some directions can not create successors. Also, we need to keep track of the visited states to prevent unnecessary computations. This will lead to decrement in the number of successor states.

1.3 Initial State

The initial state of the search algorithm is the input grid. The initial board with the agent is equivalent to the initial state of the search algorithm. The initial state will consist of S, X, and 0 symbols their respective meaning are explained in the above sections.

1.4 Goal

The goal of the program is to minimize the total distance traveled by the agent while it colors all the cells in the maze. There can be multiple solutions where the agent colored the maze. The problem is to find the solution with the minimum cost.

1.5 Step Cost

The step cost of moving from one cell to an adjacent cell is 1. Since the agent moves in directions without stopping until it reaches a wall or border, the cost of moving in a direction is between 1 to the length of to dimension - 1.

It can be calculated by the following formula.

- rp and cp stands for the position indices of the agent in the predecessor state.
- r and c stands for the position indices of the current state.
- current.cost = predecessor.cost + abs(r-rp) + abs(c-cp)
- $1 \leq abs(r - rp) + abs(c - cp) \leq len\_dimension - 1$

2 A* Search for Color-Maze

To this point, the color maze problem is converted into a search problem. The states, successors, actions, goals, and the step cost function are known. A* Search algorithm will be used to solve this search problem. Two heuristic functions will be used one is inadmissible h1 and the other one is admissible h2.

2.1 Inadmissible heuristic function h1

h1 = uncolored_cells * (wall_cells / max(row, col)) The uncolored_cells denotes the number of remaining uncolored cells. The wall_cells denotes the number of walls in the grid. The max(row, col) is the maximum dimension length.

2.1.1 Logic Behind the h1

The reason we multiply uncolored_cells with the "wall_cells / max(row, col)" is comes from the following logic:

- The "wall_cells / max(row, col)" denotes the average number of walls for the rows (Assume that row length is bigger than column length).
- To color a remaining cell in a row, one may need to exit that row "average number of walls" times.
- Each exit from the row will add unnecessary costs.
- If the agent exits the row for each remaining cell "average number of walls" times, the number of remaining cells and the average number of walls should be multiplied.

2.1.2 Why h1 is inadmissible?

In this section, a counter-example will be given to provide the inadmissibility of the h1 heuristic.

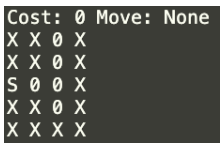

Figure 1: Initial

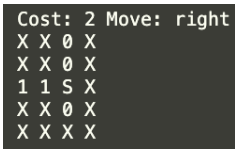

Figure 2: Step 1

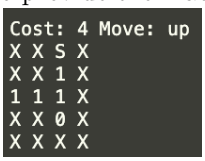

Figure 3: Step 2

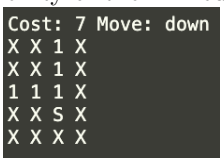

Figure 4: Step 3

Let's examine the initial state of the program. Number of uncolored cells is 5, wall-dimension ratio is 14/5 = 2.8. So our heuristic will estimate the cost as $5 * 2.8 = 14$. However, the optimal cost of this program is 6. The optimal path is right, down, and up. Even the initial state overestimates the cost with the heuristic. Also, as it can be seen h1 algorithm finds the path with 7 costs. As mentioned, the optimal cost is 6. So from these examples, the h1 is not optimal.

2.2 Admissible heuristic function h2

The admissible heuristic function is the remaining number of uncolored cells, **h2 = count(uncolored cells).** Intuitively the cost of coloring the remaining cells cannot be smaller than the number of uncolored cells since each cell has a different index in the grid. The step cost function is already explained from one cell to another cell it cannot be smaller than the Manhattan distance. As a result, h2 is intuitively admissible.

Proof by Induction:

- Let C denote the cost function for one node to another node.
- B is the function that returns the number of uncolored nodes in a path.

Statement: $\forall$ node n1 and n2, B(n1 $\rightarrow$ n2) $\leq$ C(n1 $\rightarrow$ n2). Meaning that the cost of traveling from one node to another is greater than the number of uncolored nodes in that path.

Base Case: if n1 and n2 are adjacent If they are adjacent then the agent currently is located on n1 and it will be located at the n2 at the end of the path. The cost is equal to 1. B(n1 $\rightarrow$ n2) can be 0 or 1 depending on the color of n2. If n2 is uncolored it is 1, otherwise 0. C(n1 $\rightarrow$ n2) $\geq$ B(n1 $\rightarrow$ n2). Our statement holds.

Induction Hypothesis: Assume that C(n1 $\rightarrow$ n2) $\geq$ B(n1 $\rightarrow$ n2) is true, where the path length = k $\leq$ n. Using this prove for n3, n4 where the path length is n+1: C(n3 $\rightarrow$ n4) $\geq$ B(n3 $\rightarrow$ n4).

Inductive Step: From the nodes n1 to n2 there is a path of length n+1. Let's say there is a node n3 whose path from n1 is equal to length n and path n3 to n2 is 1. n1 —n—> n3 —1–> n2

- From the inductive hypothesis: C(n1 $\rightarrow$ n3) $\geq$ B(n1 $\rightarrow$ n3)
- From the base case: C(n3 $\rightarrow$ n2) $\geq$ B(n3 $\rightarrow$ n2)
- C(n1 $\rightarrow$ n3) + C(n3 $\rightarrow$ n2) = C(n1 $\rightarrow$ n2) $\geq$ B(n1 $\rightarrow$ n3) + B(n3 $\rightarrow$ n2) = B(n1 $\rightarrow$ n2)

The above proof shows that the cost of traveling on a path is greater than the number of uncolored cells.

Let's say there are two states S1 and S2 where S2 is the goal state and S1 is any state. Let's say the n1 $\rightarrow$ n2 is the path that the agent followed in this transition.

- x1 = h2(S1), number of uncolored nodes in S1.
- x1 = B(n1 $\rightarrow$ n2) $\leq$ C(n1 $\rightarrow$ n2).

From any state to the goal state B() function and h2 return the same value which is smaller than the actual cost.

2.3 Is h2 monotonic? h(n1) $\leq$ c(n1 $\rightarrow$ n2) + h(n2)

From the above proof only the last part will change.
Let's say there are two states S1 and S2 where S2 is one of the successors of S1 (It does not have to be a direct successor). Let's say the n1 $\rightarrow$ n2 is the path that the agent followed in this transition.

- x1 = h2(S1), number of uncolored nodes in S1.
- x2 = h2(S2), number of uncolored nodes in S2.
- x1-x2 = B(n1 $\rightarrow$ n2) (since it is the number of uncolored nodes in the path) $\leq$ C(n1 $\rightarrow$ n2).
- x1 $\leq$ C(n1 $\rightarrow$ n2) + x2.
- h2(s1) $\leq$ C(n1 $\rightarrow$ n2) + h2(s2).

So, the h2 function is monotonic.

3 Results & Discussion

3.1 Discussion

3.1.1 Complexity of A* Search

I arranged the difficulty in terms of the number of possible paths. The paths are created by removing the walls. With each path new branches became available and the search algorithm needed more time and space to find the optimal solution. As can be seen from the table, the time and the space (the number of states that were explored) drastically increased as the level of the test increased. This is also expected since A* Search is based on the principles of BFS. As a result, it has an exponential time and space complexity.

3.1.2 H1 vs H2, Admissibility vs Inadmissibility

As the difficulty increases the h2 function needs more space and time compared to h1. This is visible from the table representation. Specifically for the hard test instances the amount of space (number of states) and time needed by the h2 A* Search algorithm is much more than the h1.

This difference between h1 and h2 is what is expected. The h1 is an inadmissible heuristic function that misses many branches. This branch misses in h1 leading to the overestimation of the result. On the other hand, h2 which is a monotonic function searches all necessary branches to find the optimal solution. With optimal path comes excessive calculation which results in more time and space costs.

3.2 Experimental Results Table and Graphical Representation

Tests	Grid	Difficulty Level	Cost (Distance) h1	Cost (Distance) h2	States h1	States h2	Time (h_1)	Time h2	Mem Cons h1	Mem Cons h2
1	12x12	Easy	83	83	28	28	0.00588	0.0046	79368	75888
2	12x12	Easy	134	95	143	139	0.0774	0.1690	372312	715416
3	12x12	Easy	112	112	35	82	0.01845	0.22221	91480	809728
4	12x12	Easy	120	90	37	49	0.00688	0.01723	96832	203880
5	12x12	Easy	96	96	37	42	0.00634	0.01649	96840	174592
6	12x12	Medium	123	74	41	357	0.00869	0.28699	110736	935608
7	12x12	Medium	91	78	43	297	0.00814	0.23404	112384	774432
8	12x12	Medium	121	121	39	613	0.00831	0.66426	104472	1614416
9	12x12	Medium	105	95	239	552	0.12026	0.46450	531104	1418928
10	12x12	Medium	139	105	87	879	0.03216	2.08026	227024	2257568
11	12x12	Hard	117	106	523	2878	0.63395	23.1135	1366976	7416440
12	12x12	Hard	136	109	68	809	0.01985	1.20991	178048	2094888
13	12x12	Hard	133	105	174	2761	0.09542	8.68079	450040	7137576
14	12x12	Hard	156	101	53	1803	0.01120	6.07316	138656	4701688
15	12x12	Hard	145	118	73	2567	0.02209	11.4854	188656	6454184

Table 1: Heuristic Functions' Results

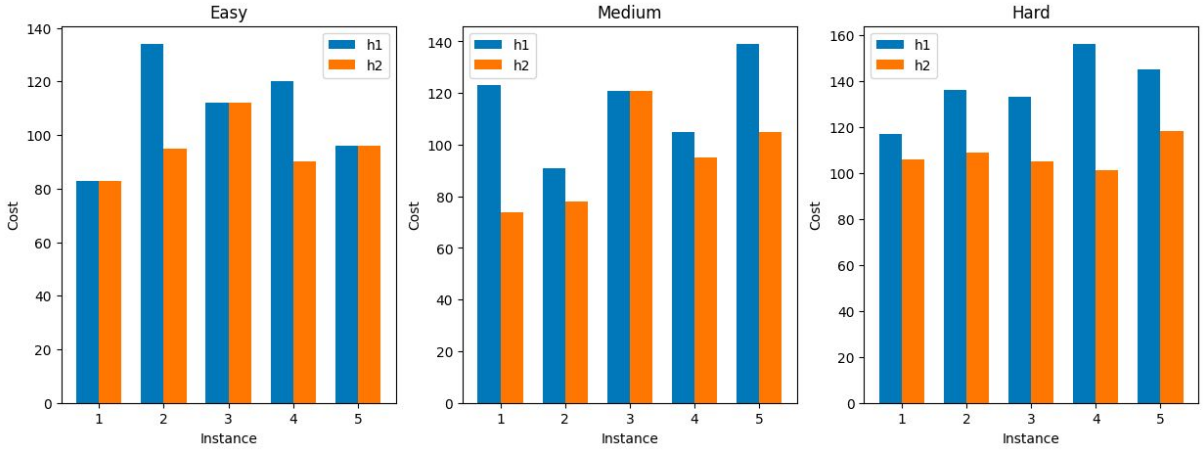

Figure 5: Cost Comparison Graphic

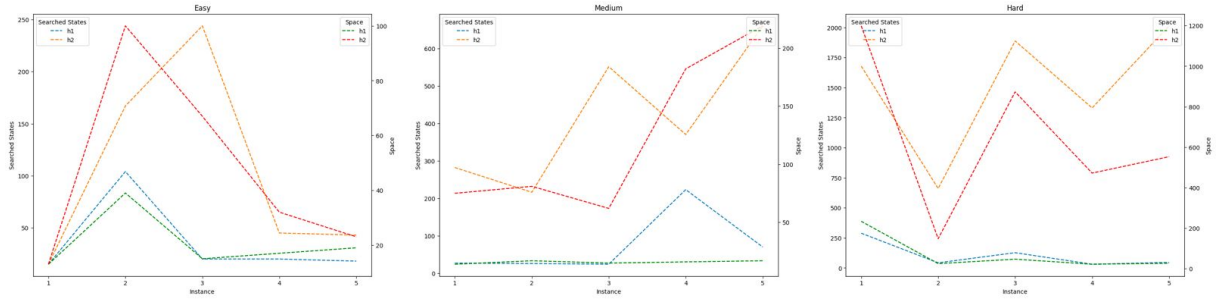

Figure 6: State Graphic