COMP30024 Artificial Intelligence by Ruipu Cui and Yuhao Liu

Part 1: Searching Algorithms

We implemented an A* search for this project. The search process can be summarized to 3 steps: 1.pop from the priority queue based on the f (= g + h) value. 2.expand form the node popped. 3.push all possible expansions into the priority queue. Iterating those 3 steps until either the goal state is reached, or the queue is empty which means no possible solution.

Specific strategy:

1.Node class: we packed each action into a Node class. Node class has the current board state and the parent Node that the action expanded from. The board state can ensure that when we pop an action that expanded from a previous action, we still have the past board information so that we don't need to remove actions from the board. The parent Node is used to retrieve backward when the goal state is reached to get the path.

2. Expansion: Use recursion to develop an empty space that adjacent to the red blocks. Stop developing when the length equals to four.

Part 2: Heuristic Function

The heuristic function contains two parts. The first part is the Manhattan distance to one of the missing blocks and the estimate number of move that need to fill the row or column corresponding to the missing blocks. We iterate each missing blocks and find the smallest sum of the two parts. Then calculate the smallest sum of each coordinate in the action. The heuristic function then returns the smallest sum among the 4 coordinates.

Specific strategy:

1. Manhattan distance: Minus the Manhattan distance by one (minimum zero) since we only need to reach the neighboring coordinate to fill the row or column of the target.

2. Estimate number of move to fill the target row or column: For each continuing empty blocks, divide the length by 4 and the round up to the smallest integer that bigger or equal to the result. Then calculate the sum.

For Figure 1, the Manhattan distance is 0 because in the very next move, we can start to fill the target's column. The estimate number of move to fill the target column of the above action is 2. Since the length of the continuing empty blocks is 8 and 8 / 4 then round up is 2.

For Figure 2, the Manhattan distance is 0. However, the estimate number of move to fill the target's column is 3. Since the column has two continuing empty blocks, one has length of one and one has length of 6. The sum is ceil(1/4) + ceil(6/4) which is 3. Therefore, the first action has a smaller heuristic value.

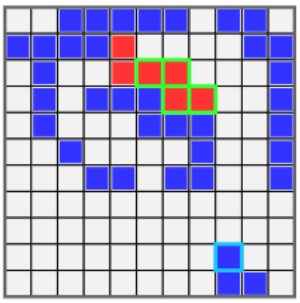


Figure 1

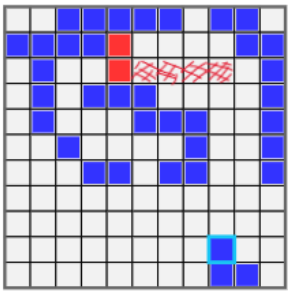


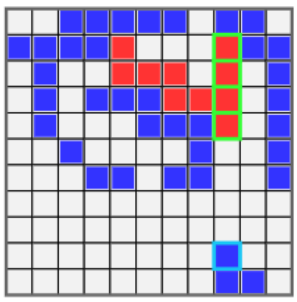
Figure 2

Part 3: Strategy We Tried and Summary

We tried to expand only one coordinate at one time and pack 4 coordinates expanded as one action. Here are pros and cons of the strategy we used and the strategy we tried in brief.

Currently used	tried
high branch factor	low branch factor (maximum 3)
low depth	high depth
Works better when solution is complicated	Works better when solution is naive
Easy to implement	Hard to implement

Figure 3



The reason that why the current strategy used works better when solution is complicated is that expand 4 coordinates at one time provide more information especially when starting to fill the target's row or column. The strategy we used can easily choose the action at figure 3 as this action didn't break the column into many parts so the action ends up in a smaller estimate cost to fill the column then ends up a smaller h value. However, if we only expand one coordinate at one time, it takes more time to come up the optimal solution as it cannot "foreseen" what possible action it can take.

In summary, out strategy has a worst-case time complexity of b^d and worst-case time complexity of b^d occurs when the algorithm needs to explore all possible paths. Our actual time and space complexity are significantly better than the worst-case scenario due to out heuristic function.

Part 4: Imagine that all Blue tokens had to be removed

Modified heuristic function: For each blue token, find the h value use the unmodified heuristic function. Then return the smallest among all blue tokens. The time complexity of calculating h value will be significantly increase since now we need to run the old heuristic function n (number of blue tokens) times. The overall time and space complexity will significantly be increased as the search depth is increased, more nodes need to be stored and higher branch factor as more tokens are cleared.