CSE 415 Assignment 2 Report: Evaluating Search Algorithms and Heuristics

Your Names Here

January 2025

1 Introduction

I am Zhehao Li. This is my report for Assignment 2 covering both blind search algorithms and heuristic search.

2 Report on Part A: Problem Formulation and Blind Search Algorithms

2.1 Part A Step 4 (c)

```
if h_remaining > 0 and h_remaining < r_remaining: return False
```

The fourth return False line in the can_move method ensures the legality of each move by preventing situations where humans are outnumbered by robots on the current side of the river after the ferry departs.

If the number of remaining humans is smaller than the number of remaining robots, which is forbidden, then this code will return False to the can_move method and this operation will not be recorded.

2.2 Part A Step 8

(your answer for the question in Part A step 8 goes into the table below, as well as the path details in 2.3 and explanations in 2.4.)

The paths are not required in the report for the entries marked ”skip.”

Problem and Algorithm	Path Found	Path length	#Nodes Expanded
Humans, Robots and Ferry / DFS	(skip)	9	10
Humans, Robots and Ferry / BreadthFS		7	10
Farmer, Fox, Chicken and Grain/ DFS		7	7
Farmer, Fox, Chicken and Grain/ BreadthFS		7	9
4-Disk Towers of Hanoi/DFS	(skip)	40	40
4-Disk Towers of Hanoi/BreadthFS		15	70

2.3 Part A Step 8, Path details

Paths found (if not shown in the table). Copy the state sequences obtained from the search algorithm on the requested problems.

- HRF/BreadthFS:

```
H on left:3
R on left:3
  H on right:0
  R on right:0
ferry is on the left.

H on left:2
R on left:2
  H on right:1
  R on right:1
ferry is on the right.

H on left:3
R on left:2
  H on right:0
  R on right:1
ferry is on the left.

H on left:0
R on left:2
  H on right:3
  R on right:1
ferry is on the right.
```

```
 H on left:2
 R on left:2
   H on right:1
   R on right:1
 ferry is on the left.

 H on left:0
 R on left:1
   H on right:3
   R on right:2
 ferry is on the right.

 H on left:1
 R on left:1
   H on right:2
   R on right:2
 ferry is on the left.

 H on left:0
 R on left:0
   H on right:3
   R on right:3
 ferry is on the right.
```

- FFCG/DFS:

```
Left bank: farmer, fox, chicken, grain
Right bank:

Left bank: fox, grain
Right bank: farmer, chicken

Left bank: farmer, fox, grain
Right bank: chicken

Left bank: grain
Right bank: farmer, fox, chicken

Left bank: farmer, chicken, grain
Right bank: fox

Left bank: chicken
Right bank: farmer, fox, grain

Left bank: farmer, chicken
Right bank: fox, grain
```

```
Left bank:
Right bank: farmer, fox, chicken, grain
```

- FFCG/BreadthFS:

```
Left bank: farmer, fox, chicken, grain
Right bank:

Left bank: fox, grain
Right bank: farmer, chicken

Left bank: farmer, fox, grain
Right bank: chicken

Left bank: grain
Right bank: farmer, fox, chicken

Left bank: farmer, chicken, grain
Right bank: fox

Left bank: chicken
Right bank: farmer, fox, grain

Left bank: farmer, chicken
Right bank: fox, grain

Left bank:
Right bank: farmer, fox, chicken, grain
```

- 4-Disk TOH/BreadthFS:

```
[[4, 3, 2, 1] ,[] ,[]]
[[4, 3, 2] ,[1] ,[]]
[[4, 3] ,[1] ,[2]]
[[4, 3] ,[] ,[2, 1]]
[[4] ,[3] ,[2, 1]]
[[4, 1] ,[3] ,[2]]
[[4, 1] ,[3, 2] ,[]]
[[4] ,[3, 2, 1] ,[]]
[[] ,[3, 2, 1] ,[4]]
[[] ,[3, 2] ,[4, 1]]
[[2] ,[3] ,[4, 1]]
[[2, 1] ,[3] ,[4]]
[[2, 1] ,[] ,[4, 3]]
[[2] ,[1] ,[4, 3]]
```

```
[[] ,[1] ,[4, 3, 2]]
[[] ,[] ,[4, 3, 2, 1]]
```

2.4 Part A Step 8, Explanations of Certain Differences, Using Towers-of-Hanoi

(i. Why the maximum length of the OPEN list is more for one algorithm than the other)

In BFS, the algorithm explores all possible states layer by layer, which causes the queue to accumulate more states. As a result, the maximum length of the OPEN list is larger.

In contrast, DFS uses a stack, focusing on diving deeper into one branch of the search tree before backtracking to upper branches. Therefore, fewer states coexist in the OPEN list at any given time.

(ii. Why why the solution PATH length is different for one algorithm from that of the other.)

BFS explores all possible paths in a breadth-first manner, ensuring the shortest path is found. In the Towers of Hanoi problem, the shortest solution path is 15 steps.

On the other hand, DFS explores paths in a depth-first manner, which may lead to exploring longer paths before backtracking to find the goal. Consequently, DFS can result in a longer path (40 steps in this case), which is not guaranteed to be optimal.

3 Report on Part B: Heuristics for the Eight Puzzle

(Your results for Part B should be reported in the table below.)

3.1 Results with Heuristics for the Eight Puzzle

Puzzle	Heuristic	Solved?	# Soln Edges	Soln Cost	# Expanded	Max Open
A	none (UCS)	Y	7	7	166	101
A	Hamming	Y	7	7	7	6
A	Manhattan	Y	7	7	7	6
B	none (UCS)	Y	12	12	1490	898
B	Hamming	Y	12	12	94	72
B	Manhattan	Y	12	12	33	25
C	none (UCS)	Y	14	14	4070	2290
C	Hamming	Y	14	14	189	127
C	Manhattan	Y	14	14	56	39
D	none (UCS)	Y	16	16	7982	4700
D	Hamming	Y	16	16	589	368
D	Manhattan	Y	16	16	148	96

```
Puzzle A: [3,0,1,6,4,2,7,8,5]
Puzzle B: [3,1,2,6,8,7,5,4,0]
Puzzle C: [4,5,0,1,2,8,3,7,6]
Puzzle D: [0,8,2,1,7,4,3,6,5]
```

3.2 (Optional) Evaluating Our Custom Heuristics

Describe your custom heuristic here. What is the underlying intuition for it? Is it admissible? Why or why not, or why is it difficult to determine if that is the case. How would you compare its computational cost with that of the Hamming heuristic and the Manhattan distance heuristic?

What puzzles did you try it on, and how did it compare? You may add rows to your table above to support your answer about comparison. (Give your heuristic an appropriate short name to identify it in the table.)

```
def h(state):
    """
    Compute Manhattan Distance first, and add extra distance.
    """
    goal_positions = {
        0: (0, 0), 1: (0, 1), 2: (0, 2),
        3: (1, 0), 4: (1, 1), 5: (1, 2),
        6: (2, 0), 7: (2, 1), 8: (2, 2)
    }
    distance = 0
    blank_pos = None

    # Compute Manhattan distance and record blank position
    for i in range(3):
        for j in range(3):
            tile = state.b[i][j]
            if tile != 0:
                goal_i, goal_j = goal_positions[tile]
                distance += abs(i - goal_i) + abs(j - goal_j)

            if state.b[i][j] == 0:
                blank_pos = (i, j)

    if blank_pos:
        blank_i, blank_j = blank_pos
        adjacent_positions = [
            (blank_i - 1, blank_j),  # Up
            (blank_i + 1, blank_j),  # Down
            (blank_i, blank_j - 1),  # Left
            (blank_i, blank_j + 1)   # Right
        ]
        """
        If the target position of a number lies in the direction of the blank tile (row or c
        an additional cost of 1 is added because the blank tile may require more moves
        to get certain numbers reach their target positions.
        """
        for adj_i, adj_j in adjacent_positions:
            if 0 <= adj_i < 3 and 0 <= adj_j < 3:
                adjacent_tile = state.b[adj_i][adj_j]
                if adjacent_tile != 0:
                    goal_i, goal_j = goal_positions[adjacent_tile]
                    if (adj_i, adj_j) != (goal_i, goal_j):
                        if (goal_i == blank_i and abs(goal_j - blank_j) == 1) or \
                                (goal_j == blank_j and abs(goal_i - blank_i) == 1):
                            distance += 1

    return distance
```

This heuristic assumes that tiles adjacent to the blank space may face additional movement constraints depending on the direction they need to move. This adjustment aims to better approximate the true cost of solving the puzzle by considering interactions between the blank tile and adjacent tiles.

If the target position of a number lies in the direction of the blank tile (row or column), an additional cost of 1 is added because the blank tile may require more moves to get certain numbers reach their target positions.

2		**1**
3	**4**	**5**
6	**7**	**8**

For example, in this state, tile 1 and tile 2 are not in their position. Its Manhattan Distance should be 3. In my heuristic, another 1 distance will be added to Manhattan Distance, because the correct position of tile 2(adjacent to the blank space) is in the direction of the blank tile. So h=4.

I think it is admissible. Although it adds some distances to Manhattan Distance, there is a cap that. In the worst case, that all 4 numbers adjacent to the blank tile have their correct position in the direction of the blank tile, the distance will be added by 4. That means this heuristic domains Manhattan Distance. So it should be admissible.

I tried it on Puzzle A,B,C,D and the results are as follows. There are improvements in both number of states expanded and max open length. So this new heuristic works more efficiently than Manhattan Diatance.

Puzzle	Heuristic	Solved?	Soln Edges	Soln Cost	Expanded	Max Open
A	Manhattan	Y	7	7	7	6
A	ManhattanBlank	Y	7	7	7	6
B	Manhattan	Y	12	12	33	25
B	ManhattanBlank	Y	12	12	23	20
C	Manhattan	Y	14	14	56	39
C	ManhattanBlank	Y	14	14	44	32
D	Manhattan	Y	16	16	148	96
D	ManhattanBlank	Y	16	16	104	74

4 Partnership Retrospective

4.1 Partnership?

Did you work in a partnership? (yes or no). No.

If so, who were the partners (repeating your names from below the title on the first page)?

4.2 Collaboration

Also if so, how did you did you divide the work of this assignment?

4.3 Newness of the Collaboration

If this was a new sort of experience for either of you, please mention that, and in what way(s) it felt new.