Assignment 1: Search Report

Aisen Kim (Chung Hyun Kim)

CSE 352 Spring 2021

Problem Formulation

TileProblem class is an abstract template that defines 8-puzzle and 15-puzzle. It clearly defines state, action, transition function, and goal test. The state of the problem is the board, which is constructed in 2D list. For the 8-puzzle problem, three by three boards are constructed and four by four boards for the 15-puzzle problem. Board contains numbers from one to n where n equals 8 or 15. The action is the movement of the blank spaces in the board. The move is defined as a list consisting of four lists of coordinates to move up, down, left, or right. Transition function alters the board with the defined action (under available movements at each tile) and returns altered state. Goal state is also defined as a method to check if current state is equal to the goal state.

Heuristics

Two heuristics used were manhattan distance and hamming distance. Manhattan distance is a summation of the distance between misplaced tile and its correct position on the board. To calculate the distance between the misplaced tile and the correct position, abs(x1 - x2) + abs(y1 - y2). This heuristic is always consistent because the estimate h value from the current node to the cost is always less than or equal to the neighbouring nodes plus the step cost. For example, in the 8-tile problem, it calculates the distance between current tile to the correct position of the tile and it also calculates all the other tiles to its correct position and adds them up. From there, the lowest sum of distances is always selected.

Second heuristic (h2) used was hamming distance. Simply, hamming distance is the number of tiles that are misplaced in the current state. Hamming distance is also consistent in that it always chooses the node(board) with the lowest number of h(value) compared to the neighbouring nodes.

Manhattan distance (h1) always dominates Hamming distance (h2) because the lowest h value that manhattan distance can have is when all tiles are just one place

away from its goal, which is the same as the highest h value that h2 can have. Therefore, h1 is said to dominate h2.

Memory Issue With A*

A start algorithm is known to expand every node that satisfies f(n) < C*(true cost to the goal state). With good heuristics, the use of memory can be decreased but it still takes up a lot of memory in the worst case O(b^d). A star algorithm explores exponentially many nodes that satisfy f(n) < C*. A problem could be reached, especially solving the 15-puzzle because it stores all the visited tiles and in the worst case, it could visit the wrong path everytime storing an exponential amount of tiles. To solve the 15-puzzle, O(b^n) is needed in the worst case.

Memory Bounded Algorithm: RBFS

Memory Bounded Algorithm was implemented using the Recursive Best First Search. It is very similar to the Best First Search but the difference is that it only uses a linear amount of space. The special part about RBFS is that it doesn't just explore continuously into the depth but stops exploring at a certain moment when it finds a worse f(n) value compared to the alternative best node that was set previously. When this happens, it moves back up to the alternative best node and starts exploring it. When it moves back up the tree, it records the node with the best f value of its child so that it can be used later when exploring. This algorithm is much more efficient in terms of space complexity. It has space complexity of O(bd) which is linear space. It is better compared to a star algorithm. RBFS is both optimal and complete. It is complete with finite state. It is optimal as long as the heuristic function satisfies some constraints. When it comes to the time complexity, it is more dependent on the heuristic function. In terms of space complexity, it uses little space (linear) but it is also incapable of taking advantage of more memory when it is given. One downside compared to A* algorithm that was noticed while implementing was that it generates node over and over.

Table on 5 Test Cases

A * with manhattan distance

File	# state explored	output	time(ms)	depth
puzzle1.txt	2	R,R	0	2
puzzle2.txt	7	U,R,D,D	0	4
puzzle3.txt	45	L,U,U,R,D,L,D,R	4	8
puzzle4.txt	28	L,L,D,R,R,R	3	6
puzzle5.txt	614	R,R,D,L,U,R,D,R,D,D	130	10

A * with hamming distance

File	# state explored	output	time(ms)	depth
puzzle1.txt	2	R,R	0	2
puzzle2.txt	4	U,R,D,D	0	4
puzzle3.txt	198	L,U,U,R,D,L,D,R	27	8
puzzle4.txt	6	L,L,D,R,R,R	0	6
puzzle5.txt	874	R,R,D,L,U,R,D,R,D,D	203	10

RBFS with manhattan distance

File	# state explored	output	time(ms)	depth
puzzle1.txt	3	R,R	0	2
puzzle2.txt	5	U,R,D,D	0	4
puzzle3.txt	16	L,U,U,R,D,L,D,R	4	8
puzzle4.txt	13	L,L,D,R,R,U,D,R	2	8
puzzle5.txt	17	R,R,D,L,U,R,D,R,U,D,D,D	7	12

RBFS with hamming distance

File	# state explored	output	time(ms)	depth
puzzle1.txt	3	R,R	0	2
puzzle2.txt	5	U,R,D,D	0	4
puzzle3.txt	15	L,U,U,R,D,L,D,R	4	8
puzzle4.txt	7	L,L,D,R,R,R	0	6
puzzle5.txt	16	R,R,D,L,U,R,D,R,D,D	3	10