Jessica Lin

CSE 352 – AI

Assignment 1 – Search Report

1. Problem Description

The TileProblem class was created in order to define the search problem. This class contains the necessary variables and functions in order to reach the goal of a problem.

1.1. Instance Variables

The TileProblem class contains seven variables: state, prevState, prevAction, emptyCell, h, g, and f.

state is a nxn array that contains the current state of the problem.

prevAction is the action taken to reach *state*.

prevState is a nxn array that contains the previous state of the puzzle before *prevAction* was performed.

emptyCell is a two-element array that contains the coordinates of the empty tile in *state.*

h, *g*, and *f*, are *h(n)*, *g(n)*, and *f-value* respectively.

1.2. Functions

The TileProblem class contains five functions: __init__, actions, transition, goalState, __eq__, and __gt__.

__init__() takes in one required argument, *new*, and two default arguments, *current*, and *action*, and creates a new TileProblem object. The *new* argument is set as the TileProblem's *state*, *current* as *prevState*, and *action* as *prevAction*. The *emptyCell* variable is then calculated and the *h*, *g*, and *f* variables set to 0. If no value for *current* is provided, then the default value is None. Similarly, the default value for *action* is "".

actions() takes in no arguments and uses *state* and *emptyCell* to return a list of all the valid actions ('U', 'D', 'L', 'R') that can be taken.

transition() takes in a single argument, *move*, which is the move to be performed on the current state of the problem. A copy of the current state is created, and the action is performed on the copy. Finally, the copy is returned.

goalState() takes in no arguments and returns True if the current state matches the goal state, and False otherwise.

__eq__() takes in one argument, *other*, which is a TileProblem object. This function is used to determine equality for the TileProblem class, where two TileProblem objects are equal if the sum of *g* and *f* are equal.

__gt__() takes in one argument, *other*, which is a TileProblem object. This function performs similar to the __eq__ function, where is it used to determine if one of two TileProblem objects is greater than the other. One TileProblem class is greater than the other is the sum of g and f is greater than the other.

2. Heuristics

The two heuristics used for the A* algorithm was the Total Manhattan Distance (h_1) and the Hamming Distance (h_2).

2.1. Total Manhattan Distance

The total Manhattan Distance is the sum of all distances between where a tile is currently, and the correct placement of the tile (excluding the empty tile). Given a tile in coordinates $[i_1, j_1]$ and its correct coordinate, $[i_1, j_1]$, the distance between the two is the sum of the difference between the i and j coordinates. In other words, $|i_1 - i_2| + |j_1 - j_2|$.

This is a consistent heuristic because each estimated distance is the direct distance from the current tile placement to the correct tile placement. In other words, the cost of h(n) is the minimum amount of moves all the tiles must move in order to reach the goal state. The step cost to reach the next state from the current state is 1. Furthermore, the only case in which $h(n') < h(n)$ is when the tile that is swapped with the empty space gets closer to its correct location. However, since that tile only moves one tile distance, then that tile can only get closer by a tile distance of 1. Thus, $h(n') \geq h(n) - 1$.

In order for a heuristic to be consistent, the following inequality must hold true: $h(n) \leq c(n,n') + h(n')$. Let us consider $h(n')$ to be the minimum possible value ($h(n) - 1$). Thus, $c(n,n') + h(n') = 1 + h(n) - 1 = h(n)$, where the inequality still holds true.

2.2. Hamming Distance

The Hamming Distance is the number of misplaced tiles in the problem (excluding the empty tile). To determine how many misplaced tiles there are, the state is compared to the goal. For every tile in the state that does not match the corresponding tile in goal, the number of misplaced tiles is increased.

This is a consistent heuristic because each estimated distance is the total number of misplaced tiles. In other words, the cost of h(n) is how many tiles that are not in the right place. The step cost to reach the next state from the current state is 1. Furthermore, the only case in which $h(n') < h(n)$ is when the tile that is swapped with the empty space reaches its destination. However, since only one tile is swapped, then only one more tile can be placed in its correct location. Thus, $h(n') \geq h(n) - 1$.

In order for a heuristic to be consistent, the following inequality must hold true: $h(n) \leq c(n,n') + h(n')$. Let us consider $h(n')$ to be the minimum possible value ($h(n) - 1$). Thus, $c(n,n') + h(n') = 1 + h(n) - 1 =$ h(n), where the inequality still holds true.

2.3. Dominance

$h_1(n) > h_2(n)$ (Manhattan Distance dominates Hamming Distance) for all n. This is because the Total Manhattan Distance takes into consideration the distance a tile is from its correct placement. In other words, the cost of $h_1(n)$ is the minimum amount of moves all the tiles must move in order to reach the goal. For Hamming distance, it only takes into consideration whether or not a tile is misplaced. That means the max value of $h_2(n)$ is the total number of tiles. Thus, it can clearly be seen that for all n, $h_1(n) > h_2(n)$.

3. Memory issue with A*

The memory issue that comes up with the A* algorithm is that as the number of explored states increase, the memory usage of this algorithm will also increase exponentially. This happens because for each TileProblem state explored, all the children created from each valid action is stored into the frontier. Furthermore, all the states that have already been explored are also kept in memory.

While more memory is used to solve a complex 8-puzzle, the memory problem with the A* algorithm really comes in with the 15-puzzle. This is because not only would each TileProblem require more space on their own, but there are also more possible moves to reach the goal. Given that there's more tiles, there will be even more states to be explored, and thus, more children created and added to the frontier as well.

4. Memory-bounded Algorithm

The memory-bounded algorithm, RBFS, is a recursive best-first search algorithm. At any node it is currently at, it keeps track of what the best alternative path to the goal is. If at any point the current path cost has exceeded that of the alternative path, then the algorithm recurse back to the alternative path.

This addresses the memory issue with A* graph search because instead of keeping all the created TileProblem nodes in memory, nodes are removed from memory when a certain path is not optimal. In other words, when the algorithm recurses back to the alternative path from the current node, the nodes along the that are removed from memory, thus using linear space.

This algorithm is complete because for each node that's being explored, all the possible states that can be produced with valid actions are created. From this, all the states are explored in order of least cost. If a node explored doesn't lead to a solution, then the next best state is then explored.

This algorithm is optimal because the optimal path (best alternative path) is always kept track of. If at any point the current node being explored costs more than the alternative, then the algorithm will not continue with that state and instead, recurse back to the alternative.

5. Results on 5 test files

	Puzzle 1	Puzzle 2	Puzzle 3	Puzzle 4	Puzzle 5
h_1 Output	R,R	U,R,D,D	L,U,U,R, D,L,D,R	L,L,D, R,R,R	R,D,R,U,L ,D,R,R,D,D
h_2 Output	R,R	U,R,D,D	L,U,U,R, D,L,D,R	L,L,D, R,R,R	R,R,D,L,U, R,D,R,D,D
h_1 States Explored	3	5	16	7	22
h_2 States Explored	3	5	27	7	27
h_1 Time (ms)	1.03	1.89	4.06	1.00	5.11
h_2 Time	1.00	0.32	3.03	0.01	4.86
h_1 Depth	2	4	8	6	10
h_2 Depth	2	4	8	6	10

6. Citations

[1] A* algorithm: https://piazza.com/class_profile/get_resource/kkkgroo574w5gb/kkzxegpfusxkr?

[2] RBFS algorithm: https://pages.mtu.edu/~nilufer/classes/cs5811/2012-fall/lecture-slides/cs5811-ch03-search-b-informed-v2.pdf

[3] Heuristics: https://en.wikipedia.org/wiki/Admissible_heuristic