Problem 3 Writeup - 15 Puzzle Solution

Bhargavi Chalasani

Problem Description

Given an initial configuration as an input from the user, the goal is to provide a short set of moves to reach the following goal configuration:

1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 0

Search algorithm

Since we are trying to find the most optimal solution, this search problem can be best modeled using A* algorithm. For A* algorithm to be optimal, we need to consider a heuristic which is admissible.

The following heuristics are considered for this problem:

1. Misplaced tiles

This heuristic considers the number of misplaced tiles in the current board configuration to estimate the number of moves required to reach the goal state. Since at least the number of misplaced tiles need to be fixed before we can reach the goal state, we are never over estimating the cost with this heuristic. Hence it is admissible. Since a maximum of 3 tiles can be moved per move and 'n' number of such moves can be possible before we can reach a goal configuration, the heuristic for this case is considered as H(s) = misplaced_tiles/3

2. Manhattan distance

This heuristic considers the manhattan distance of each tile to the goal position. Since moving each tile or a set of tiles will increase or decrease the manhattan distance by a maximum of 3, this heuristic is admissible if H(s) = manhattan_distance/3

This is better than the misplaced tiles since it accounts for tile positions.

3. Manhattan distance with linear conflict

This is an addition to the above heuristic with the expectation that any tiles which are in linear conflict in a row or column will need more than the manhattan distance to reach the goal position. To account for the multi-tile moves and to be admissible for this case, I have considered this heuristic as H(s) = (manhattan_distance)/3 + (number_of_linear_conflicts)/3

4. Disjoint pattern matching

This tries to improve the above heuristic by dividing the problem to first solve subproblems. The 15 puzzle can be divided into 6-6-3 or 4-4-4-3 set of problems and pre calculate the distance of each of these configuration to the goal state for the current board for all possible configurations. The heuristic can then be defined as H(s) = d[1-2-3-4-5-6] + d[7-8-9-10-11-12] + d[13-14-15]

This heuristic was not considered in the final implementation because the cost of pre-calculating the distances for these subproblems is very high and in my opinion, is more apt where this database can be created and used as a lookup but not for this assignment where it will be recreated and will push up the runtime.

The final heuristic considered was **manhattan distance with linear conflict** which gave a more optimal solution than just manhattan distance.

This heuristic is also admissible since the cost to reach the goal state is never more than the cost to reach an intermediary state when f(s) or the evaluation function is calculated as

f(s) = g(s) + h(s)

where g(s) = cost to reach the present state + distance between the present state and its successor being considered.

Implementation

Since the heuristic is consistent, I have considered Search algorithm #3 discussed in class which maintains a list of expanded nodes and does not revisit these nodes while checking for the optimal path. The node with the highest priority or the lowest cost is popped from the fringe at each stage and checked if it is the goal state. If it is not the goal and not previous expanded, the set of successors are built by the set of valid moves. They are pushed to the fringe with their expected cost f(s). This process happens iteratively till goal state is reached.

Observation and Tweaks

In my observation, when considering the value (cost_to_reach_present_state)/3 instead of its original value as g(s) the search converges to a solution much faster and gives a more optimal solution. Since (manhattan_distance)/3 was considered as the heuristic to calculate the cost to account for multi-tile moves, scaling down the cost to go the next board state to be in line with the heuristic lets us explore the next set of successors to a board state faster and avoids a BFS style search by not going back to the very initial moves which are much farther from the goal state. For the multiple board inputs considered, this did not affect the optimality of the path.

I had considered two equations for cost evaluation: f(s) = g(s) + h(successor) – h(current_node)

and f(s) = g(s) + 2*h(successor) – h(current_node)

From A* algorithm specified in the below references, the second formulation seems to be correct but the path was not optimal since g(s) factors in the heuristic from current node in my implementation. Considered the first formulation for f(s) to ensure the cost is not over estimated.

References:
https://stackoverflow.com/questions/42901623/re-visiting-on-a-algorithm-do-i-have-to-check-open-list-or-closed-list
https://stackoverflow.com/questions/408952/correct-formulation-of-the-a-algorithm?rq=1