15-puzzle: A* Search

CS F407

Manhattan distance and Linear Conflicts

(Aayush Mathur, 2018A7PS0729G)
Instructor Incharge: Sujith Thomas

- We push the initialState/node into a PriorityQueue, which maintains the node with least f(n) at the front.
- We pop a node with least f(n) from the PriorityQueue and check if it's the goal State. If it is, then we return the path and the number of nodes generated, else we calculate f(n) for each neighboring state of the node and push them into the PriorityQueue. We do this until we get to the goal state.
- To calculate the f(n) we need g(n) and h(n). g(n) is trivial to calculate. However, for h(n), we use the Heuristic Class. This class does some pre-computations. These pre-computations are helping to find Manhattan Distances quickly but, more importantly, the Linear Conflicts.
- The Heuristic class object upon instantiation calls a function to build the conflict_value_table, which it loads from a .dat file. It then uses this table to pre-compute the number of moves to add to the estimate f(n) for all possible row and col conflicts.
- The Heuristic Class has a heuristic function that accepts a start state, returns the sum of Manhattan Distances & the moves/distances from Linear Conflicts estimate. It finds the types of linear conflicts present in the start state and then uses the pre-computed row and col conflict dictionaries to get the estimated number of moves for the particular type of conflict..

- f(n) estimates the cheapest cost solution path that goes through n.

f(n)

h(n) — h(n) = estimate of the cheapest cost of a path from n to a goal.

cost of current path from start to node n in the search tree — **g(n)**

f(n) = g(n) + h(n) — Evaluation function:

absolute horizontal distance + absolute vertical distance of every tile to its correct location — **Manhattan Distance**

Linear Conflicts — Two tiles tj and tk are in a linear conflict if tj and tk are the same line, the goal positions of tj and tk are both in that line, tj is to the right of tk, and goal position of tj is to the left of the goal position of tk.

h(n) = Manhattan Distances + Linear Conflicts

Optimal or Not?

	input01	input02	input03	input04
MD+LC	0.23secs, 32 nodes	0.60secs, 21958 nodes	2.87secs, 139319 nodes	6.10secs, 304501 nodes
MD	0.001secs, 32 nodes	1.68secs, 106515 nodes	22.03secs, 1265792 nodes	15.49secs, 855122 nodes

The above results(time & nodes generated) were obtained on a machine with CPU: Intel i7-8750H (12) @ 4.100GHz, Memory: 8667MiB / 15871MiB, OS: Ubuntu 20.04.1 LTS x86_64 for input_file1.txt - input_file4.txt

MD: Manhattan Distance, LC: Linear Conflicts

An Optimal solution has the lowest path cost among all solutions.

Since we're popping the node with the least f(n) and return it's path only if it's a goal state, we can be sure that there is no other path to the goal state having lesser cost than the path returned by the algorithm because no different path has lesser f(n).

To confirm the above, it suffices to show that f(n) is a valid measure of path cost so that it never overestimates the actual path cost.

It's trivial to show that actual path cost is no less than g(n) + Manhattan Distance. However, the Manhattan Distance is not as Informed and is relaxed because it assumes that the tiles on the board can go over each other. This problem is taken care of by adding the linear conflicts to the heuristic. Every time there is a linear conflict, we will require at least 2 more moves than the manhattan distance because one of the 2 conflicting tiles will have to go out of the row/col and then come back in again to attain the goal state.
The conflict_values_table has the exact values of all extra moves required due to different types of linear conflicts.

Algorithm used to calculate conflict_value_table dictionary stored in .dat file can be found here in Fig. 5.