CSE537 HW1

Tim Zhang

October 9, 2016

1 Heuristics

My implementation of A* used the general tree search algorithm as a base and as such I will only show the admissibility of my chosen heuristics.

1.1 Misplaced Tiles Heuristic

The misplaced tiles heuristic is derived by relaxing the formal problem definition of the $n \times n$ puzzle to allow tiles to be moved to any space regardless of if the space is empty or the relative location of the space with respect to the current location of the tile.

The misplaced tiles heuristic is admissible since in the best case a tile is misplaced by the correct single empty adjacent square or it is already in the correct position. When this occurs the actual distance the tile needs to move is equal to the estimated distance provided. That is, it is either 1 or 0 respectively.

For all other possibilities the number of moves required to correctly position the misplaced tile will be greater than the single move estimated due to the fact that the blank will have to move at least twice to be correctly adjacent to the misplaced tile. Since this is the case the heuristic will always underestimate the number of required moves to solve the puzzle and is therefore admissible.

1.2 Manhattan Distance Heuristic

The Manhattan distance is also derived by relaxing the formal definition of the problem. In this relaxation a tile may move to an adjacent position regardless of if it is occupied or not.

This heuristic is admissible since by not considering the occupancy of an adjacent tile we disregard a series of moves needed in the real problem for each Manhattan move. In the best case the adjacent tile is both empty and exactly the correct position and there is only a single misplaced tile, when this occurs the Manhattan distance is the same as the true distance. Otherwise the true distance will always be longer since it will take some series of moves to achieve the same effect.

2 Memory Issues

Even with the tree search implementation of A* the memory requirements become prohibitive in the 4×4 case. This is due to the fact that the frontier is kept in memory and nodes are not goal tested until they are considered for expansion.

The amount of memory needed to solve a solvable instance of the 15-puzzle is bounded by $O(4^d)$ where d is the depth of the optimal solution since each state has at most 4 possible successors. By ignoring reversible actions we can reduce this to $O(3^d)$ nodes which we need to keep in memory. So for each best node we consider we add 3 more nodes to the frontier in the worst case.

Empirically, a single instance of the 4×4 puzzle used up over 11GB of memory on my computer and was terminated on my Windows machine where processes have a memory limit.

3 Iterative Deepening A*

IDA* is a recursive search algorithm which is essentially an uninformed DFS which kills branches when the current node has an f-cost greater than the current depth limit. When this happens the parent of the node will keep track of the f-cost of the pruned node and continue the recursion through each of it's successors. Finally, when all successors are accounted for, if there is not a solution found the node will return the smallest f-cost which exceeded the depth limit. If a solution is found it is returned, and if no solution is found and there was no node which exceeded the limit then there is no solution on the branch. The original depth limit is initialized to the f-cost of the start state.

IDA* is both optimal and complete given that the heuristic function is admissible. This is because IDA* will search the entire space up to and including the contour where the f-values are equal to the cost of the optimal solution.

The time complexity of IDA* is $O(b^{\epsilon d})$ where d is the depth of the optimal solution, ϵ is the relative error and b is the branching factor this is because the time complexity of IDA* is bounded by the iteration which finds the optimal solution. In this iteration, in the worst case, every node will be considered and there are $b^{\epsilon d}$ nodes in total.

The space complexity of IDA* is $O(d)$ where d is the depth of the optimal solution. This is because there will be at most d nodes on a path and (recursive) IDA* will only keep a single path in memory at a time.

4 A* Performance

See section 4.3 for initial state references.

4.1 Manhattan Distance Heuristic

Initial State	States Explored	Time (ms)	Depth
1	986	48.69	20
2	1612	84.134	24
3	29271	1749.189	24
4	3326	176.70600000000002	24
5	219	12.143	20
6	2900	161.10299999999998	24
7	536	27.78300000000001	20
8	1064	56.297	20
9	2294	124.398	22
10	8362	491.947	26
11	39	1.948999999999998	14
12	2115	125.49600000000001	24
13	1133	60.93800000000001	22
14	946	47.239000000000004	20
15	30	2.2449999999999974	14
16	705	36.932	20
17	487	24.528999999999996	22
18	2206	123.22400000000002	24
19	1018	56.41400000000001	22
20	546	29.629999999999995	20

4.2 Misplaced Tiles Heuristic

Initial State	States Explored	Time (ms)	Depth
1	7493	317.771	20
2	52059	2473.1639999999998	24
3	1018590	64340.001000000004	30
4	43662	2359.032	24
5	4369	198.59499999999997	20
6	41415	2226.945	24
7	5062	234.983	20
8	6865	338.041	20
9	17671	909.362	22
10	125701	7530.735	26
11	224	9.323	14
12	40884	2108.9350000000004	24
13	18447	933.249	22
14	5720	266.027	20
15	225	10.811	14
16	5187	238.56500000000003	20
17	13525	660.457	22
18	42101	2208.1290000000004	24
19	18372	941.467	22
20	5513	255.20099999999996	20

4.3 Initial States

Shown as an list with 0 being the blank tile.
1. (5 1 6 3 0 8 7 4 2)
2. (8 5 6 7 0 4 3 1 2)
3. (0 5 4 8 6 7 2 3 1)
4. (8 1 2 7 5 4 3 6 0)
5. (4 7 0 8 6 1 5 3 2)
6. (5 6 1 8 4 2 0 3 7)
7. (8 1 0 4 7 3 5 6 2)
8. (2 7 1 4 0 5 8 3 6)
9. (6 2 3 5 1 8 7 4 0)
10. (2 7 1 6 5 8 0 4 3)
11. (0 8 5 1 6 2 4 7 3)
12. (5 4 0 7 8 1 3 2 6)
13. (6 7 2 3 0 8 4 1 5)
14. (0 2 6 5 1 3 8 4 7)
15. (6 2 0 1 4 8 7 3 5)
16. (5 2 6 1 7 4 0 8 3)
17. (0 2 7 8 6 3 5 1 4)
18. (5 3 7 8 4 6 1 2 0)
19. (6 1 8 5 0 3 2 4 7)
20. (4 8 5 2 3 6 7 1 0)