Adarsh Trivedi
atrived
200317604

1. Heuristic Function:

   The idea is to simplify the problem a bit (or relax) and then try calculating the heuristic value of every state.

   I simplify the problem by assuming that the problem only consists of only six intersection tiles [(0,0), (180,180), (90,0), (90, 180), (90, 90), (90, 270)].

   Later, the approach is to calculate the avg of the sum of shortest distance (in terms of no of rotation) of only these six tiles and take actions accordingly.

   The idea of using the shortest number of rotation instead of straight forward Manhattan distance [(x2-x1) - (y2-y1)] of the degree of movement of the tiles is because the final cost is in terms of the number of rotation to reach the goal instead of degree of movement.

   Hence, when adding g(n) which is the count of rotations till now to h(n), h(n) should again be in terms of rotation and not degree.

   Admissible (Why will this heuristic be optimal and always get a value lesser than the actual cost?)

   a. Shortest Distance: The idea of using the shortest distance average as a heuristic is to create a lower bound to the actual cost.
   b. The shortest distance will always ensure that the cost count is minimal and will always be lesser or equal (in a hypothetical case of all 6 mentioned tiles take the right position by avg rotation calculated as the heuristic value).
   c. Whenever the action is taken with this heuristic, the expanded node is in its optimal path because of the shortest distance consideration.

Since the search space is a tree, the need to show consistency is not applicable.

A possible shortcoming of this heuristic:

The initial idea of minimizing the number of tiles to six was to relax the constraints on the problem so as to avoid overestimating the heuristic value. I'm not sure if this logic actually helps to improve the heuristic or can be a cause of high underestimation, which may make the heuristic optimal but very slow to reach the goal state.

2. Code submitted as Assign2.zip and document as Assign2.pdf.

3.

The code runs with optimal results for all PathN-n.md files.

Puzzle files:

Detail Running Parameters (For puzzle 0):

Puzzle File: Puzzle-0.md

Time	BFS	A-Star	RBFS
100 seconds	States Explored: 322456 Maximum Frontier Length: 1362224 Current Path Length: 8	States Explored: 280261 Maximum frontier length: 1132001 Current Path Length:9	States Explored: 39998 Current Path Length: 7
200 seconds	States Explored: 456618 Maximum Frontier Length: 1956675 Current Path Length: 8	States Explored: 455037 Maximum frontier length: 1556872 Current Path Length:10	States Explored: 93412 Current Path Length: 8
300 seconds	States Explored: 571467 Maximum Frontier	States Explored: 547935 Maximum frontier	States Explored: 135436 Current Path Length: 8

	Length: 2452883 Current Path Length: 9	length: 1908038 Current Path Length:9	
400 seconds	States Explored: 656091 Maximum Frontier Length: 2811114 Current Path Length: 9	States Explored: 567062 Maximum frontier length: 2300019 Current Path Length: 8	States Explored: 181215 Current Path Length: 7
500 seconds	States Explored: 756981 Maximum Frontier Length: 3190874 Current Path Length: 9	States Explored: 647203 Maximum frontier length: 2509127 Current Path Length: 8	States Explored: 236132 Current Path Length: 7
600 seconds	States Explored: 785496 Maximum Frontier Length: 3367748 Current Path Length: 9	States Explored: 680670 Maximum frontier length: 2798090 Current Path Length: 8	States Explored: 267133 Current Path Length: 9
700 seconds	States Explored: 941986 Maximum frontier length: 4033704 Current Path Length: 9	States Explored: 695182 Maximum frontier length: 3004563 Current Path Length: 9	States Explored: 301232 Current Path Length: 9
800 seconds	States Explored: 942740 Maximum frontier length: 4037586	States Explored: 765369 Maximum frontier length: 3216576	States Explored: 323412 Current Path Length: 9

	Current Path Length: 9	Current Path Length: 9	
900 seconds	States Explored: 1011249 Maximum frontier length: 4329546 Current Path Length: 9	States Explored: 765369 Maximum frontier length: 3499891 Current Path Length:9	States Explored: 356989 Current Path Length: 8

Brief Report: For algorithms run over Puzzle 0,8, and 11 for 15 mins each.

Puzzle	BFS	A-Star	RBFS
Puzzle-0	Max Explored States: 1011249 Max Frontier Size: 4329546	Max Explored States: 795925 Max Frontier Size: 2981204	Max States Explored: 465817
Puzzle-8	Max Explored States: 1121980 Max Frontier Size: 4532192	Max Explored States: 809165 Max Frontier Size: 3032418	Max States Explored: 489132
Puzzle-11	Max Explored States: 1090231 Max Frontier Size: 4423217	Max Explored States: 778093 Max Frontier Size: 2899801	Max States Explored: 455392
Average Puzzle	Average States Explored: 1074486 Average Frontier Size: 4428318	Average States Explored: 794394 Average Frontier Size: 2971141	Average States Explored: 470113

Inferences:

1. Breadth-First Search is a good algorithm when the solution is not very deep in the search space. The algorithm begins saturating as the levels begin to increase.
2. A*, when compared to BFS, needs to visits far fewer nodes in the search space due to the guiding heuristics and hence achieves much faster results.
3. RBFS is a good optimization for saving space. The problem can be recursion. From my observation doing the assignment (in python) and exploring the results, the algorithms do save space but explore fewer states than A* in the same amount of time.

The best algorithm for this problem: A*

BFS is not a suitable algorithm because of its uninformed nature and the algorithm saturates after traversing few levels (in a tree space search).

A* is better than RBFS when the tradeoff between time and space is considered under the resources available under which assignment has been done and the language of choice.

Python (the language of implementation) is not suited for recursive algorithms and does not have a good reputation with recursion.

It is also evident with the observations from the detailed report. The number of explored states for RBFS is far less than states explored by A* in a given amount of time.

Hence, with space being a cheap resource and time being of more importance A* is a better algorithm for this problem.