California State Polytechnic University, Pomona

A* Search Algorithm:
with 8-Piece Puzzle

Theresa Van
011821381
CS 4200-01: Artificial Intelligence
Dominick Atanasio
02 Feb 2020

Purpose:

The purpose of this project is to implement the A* algorithm in through a graph search and a tree search. Each search will be done with two heuristic measures: h_1, which measures the amount of misplaced tiles in a given state, and h_2, which measures the distance of all the misplaced tiles from their correct location in a given state. We will be comparing the time it takes to execute each algorithm with each heuristic and the amount of nodes searched.

Implementation:

The following algorithm below is implemented for graph search for A* search.

```
function A-STAR-SEARCH (problem, heuristic) returns a solution, or failure
      node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
      frontier ← a priority queue ordered by ESTIMATED-COST with node as the
                  only element
                  /* ESTIMATED-COST = PATH-COST [g(n)] + EST-COST-TO-GOAL
                     [h(n)] */
      explored ← an empty set

      loop do
            if EMPTY?(frontier) then return failure
            node ← POP(frontier) // choose the lowest-cost node in frontier
            if problem.GOAL-TEST (node.STATE) then return SOLUTION(node)
            add node.STATE to explored
            for each action in problem.ACTIONS(node.STATE) do
                  child ← CHILD-NODE(problem, heuristic, node, action)
                  if child.STATE is not in explored or frontier then
                        frontier ← INSERT(child, frontier)
                  else if child.STATE is in frontier with higher
                              ESTIMATED-COST then
                        replace that frontier node with child
```

And the following algorithm below is implemented for tree search for A* search.

```
function A-STAR-SEARCH (problem heuristic) returns a solution, or failure
      node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
      frontier ← a priority queue ordered by ESTIMATED-COST with node as the
                  only element
                  /* ESTIMATED-COST = PATH-COST [g(n)] + EST-COST-TO-GOAL
                     [h(n)] */

      loop do
            if EMPTY?(frontier) then return failure
            node ← POP(frontier) // choose the lowest-cost node in frontier
            if problem.GOAL-TEST (node.STATE) then return SOLUTION(node)
            for each action in problem.ACTIONS(node.STATE) do
```

```
child ← CHILD-NODE(problem, heuristic, node, action)
frontier ← INSERT(child, frontier)
```

The difference between the two algorithms is the graph search version keeps track of nodes that have already been explored whereas the tree search does not, so the algorithm will iterate over nodes that may have already been explored. This makes the tree search less efficient.

Data:

Depth	h1 Average Time (ns)	h1 Average Nodes	h2 AverageTime (ns)	h2 Average Nodes
2	167149	2	2888876	2
4	55508	3	2422257	3
6	136183	5	7921515	5
8	358022	5	9006953	6
10	708638	7	16569579	8
12	2041093	7	31784592	9
14	3270504	8	48030677	9
16	7813259	9	56920453	10
18	11380447	9	71158672	11
20	25031532	10	87278095	13
22	12029947	11	56730870	14
24	13910026	10	44679320	12

The above data set is done with graph search of the A* search algorithm. 100 instances of each solution depth was tested and the average is given in the table. The average execution time and average amount of searched nodes for h1 is consistently lower than that of h2.

When executing the tree search of the A* search algorithm, an `OutOfMemoryError` is returned. This is due to the algorithm's lack of an explored set to curb the amount of nodes placed into the frontier. The tree search generates an obscene amount of child nodes to be placed into the frontier, exhausting the heap space. The tree search is able to execute when solution depths of a smaller caliber are inputted.

3 sample solutions of depth greater than 10 are given below.

```
  Search with heuristic 1.
8 6 0
1 3 4
2 5 7

  Step 1
8 6 4
1 3 0
2 5 7

  Step 2
8 6 4
1 0 3
2 5 7

  Step 3
8 0 4
1 6 3
2 5 7

  Step 4
0 8 4
1 6 3
2 5 7

  Step 5
1 8 4
0 6 3
2 5 7
```

```
  Step 6
1 8 4
6 0 3
2 5 7

  Step 7
1 8 4
6 3 0
2 5 7

  Step 8
1 8 4
6 3 7
2 5 0

  Step 9
1 8 4
6 3 7
2 0 5

  Step 10
1 8 4
6 3 7
0 2 5

  Step 11
1 8 4
0 3 7
6 2 5
```

```
  Step 12
1 8 4
3 0 7
6 2 5

  Step 13
1 8 4
3 2 7
6 0 5

  Step 14
1 8 4
3 2 7
6 5 0

  Step 15
1 8 4
3 2 0
6 5 7

  Step 16
1 8 4
3 0 2
6 5 7

  Step 17
1 0 4
3 8 2
6 5 7
```

```
  Step 18
1 4 0
3 8 2
6 5 7

  Step 19
1 4 2
3 8 0
6 5 7

  Step 20
1 4 2
3 0 8
6 5 7

  Step 21
1 4 2
3 5 8
6 0 7

  Step 22
1 4 2
3 5 8
6 7 0

  Step 23
1 4 2
3 5 0
6 7 8
```

```
  Step 24
1 4 2
3 0 5
6 7 8

  Step 25
1 0 2
3 4 5
6 7 8

  Step 26
0 1 2
3 4 5
6 7 8
```

```
  Search with heuristic 2.
8 6 0
1 3 4
2 5 7

  Step 0
8 6 4
1 3 0
2 5 7

  Step 1
8 6 4
1 0 3
2 5 7

  Step 2
8 0 4
1 6 3
2 5 7

  Step 3
0 8 4
1 6 3
2 5 7

  Step 4
1 8 4
0 6 3
2 5 7

  Step 5
1 8 4
6 0 3
2 5 7

  Step 6
1 8 4
6 3 0
2 5 7

  Step 7
1 8 4
6 3 7
2 5 0

  Step 8
1 8 4
6 3 7
2 0 5

  Step 9
1 8 4
6 3 7
0 2 5

  Step 10
1 8 4
0 3 7
6 2 5

  Step 11
1 8 4
3 0 7
6 2 5

  Step 12
1 8 4
3 2 7
6 0 5

  Step 13
1 8 4
3 2 7
6 5 0

  Step 14
1 8 4
3 2 0
6 5 7

  Step 15
1 8 4
3 0 2
6 5 7

  Step 16
1 0 4
3 8 2
6 5 7

  Step 17
1 4 0
3 8 2
6 5 7

  Step 18
1 4 2
3 8 0
6 5 7

  Step 19
1 4 2
3 0 8
6 5 7

  Step 20
1 4 2
3 5 8
6 0 7

  Step 21
1 4 2
3 5 8
6 7 0

  Step 22
1 4 2
3 5 0
6 7 8
```

```
  Step 23
1 4 2
3 0 5
6 7 8

  Step 24
1 0 2
3 4 5
6 7 8

  Step 25
0 1 2
3 4 5
6 7 8
```

```
  Search with heuristic 1.
5 8 1
2 3 0
7 4 6

  Step 1
5 8 0
2 3 1
7 4 6

  Step 2
5 0 8
2 3 1
7 4 6

  Step 3
0 5 8
2 3 1
7 4 6

  Step 4
2 5 8
0 3 1
7 4 6

  Step 5
2 5 8
3 0 1
7 4 6
```

```
  Step 6
2 5 8
3 1 0
7 4 6

  Step 7
2 5 0
3 1 8
7 4 6

  Step 8
2 0 5
3 1 8
7 4 6

  Step 9
0 2 5
3 1 8
7 4 6

  Step 10
3 2 5
0 1 8
7 4 6

  Step 11
3 2 5
7 1 8
0 4 6
```

```
  Step 12
3 2 5
7 1 8
4 0 6

  Step 13
3 2 5
7 1 8
4 6 0

  Step 14
3 2 5
7 1 0
4 6 8

  Step 15
3 2 0
7 1 5
4 6 8

  Step 16
3 0 2
7 1 5
4 6 8

  Step 17
3 1 2
7 0 5
4 6 8
```

```
  Step 18
3 1 2
0 7 5
4 6 8

  Step 19
3 1 2
4 7 5
0 6 8

  Step 20
3 1 2
4 7 5
6 0 8

  Step 21
3 1 2
4 0 5
6 7 8

  Step 22
3 1 2
0 4 5
6 7 8

  Step 23
0 1 2
3 4 5
6 7 8
```

```
  Search with heuristic 2.
5 8 1
2 3 0
7 4 6

  Step 0
5 8 0
2 3 1
7 4 6

  Step 1
5 0 8
2 3 1
7 4 6

  Step 2
0 5 8
2 3 1
7 4 6

  Step 3
2 5 8
0 3 1
7 4 6

  Step 4
2 5 8
3 0 1
7 4 6

  Step 5
2 5 8
3 1 0
7 4 6

  Step 6
2 5 0
3 1 8
7 4 6

  Step 7
2 0 5
3 1 8
7 4 6

  Step 8
0 2 5
3 1 8
7 4 6

  Step 9
3 2 5
0 1 8
7 4 6

  Step 10
3 2 5
7 1 8
0 4 6

  Step 11
3 2 5
7 1 8
4 0 6

  Step 12
3 2 5
7 1 8
4 6 0

  Step 13
3 2 5
7 1 0
4 6 8

  Step 14
3 2 0
7 1 5
4 6 8

  Step 15
3 0 2
7 1 5
4 6 8

  Step 16
3 1 2
7 0 5
4 6 8

  Step 17
3 1 2
0 7 5
4 6 8

  Step 18
3 1 2
4 7 5
0 6 8

  Step 19
3 1 2
4 7 5
6 0 8

  Step 20
3 1 2
4 0 5
6 7 8

  Step 21
3 1 2
0 4 5
6 7 8

  Step 22
0 1 2
3 4 5
6 7 8
```

```
  Search with heuristic 1.
1 8 0
5 7 2
6 4 3

  Step 1
1 8 2
5 7 0
6 4 3

  Step 2
1 8 2
5 0 7
6 4 3

  Step 3
1 8 2
0 5 7
6 4 3

  Step 4
1 8 2
6 5 7
0 4 3

  Step 5
1 8 2
6 5 7
4 0 3
```

```
  Step 6
1 8 2
6 5 7
4 3 0

  Step 7
1 8 2
6 5 0
4 3 7

  Step 8
1 8 2
6 0 5
4 3 7

  Step 9
1 8 2
6 3 5
4 0 7

  Step 10
1 8 2
6 3 5
0 4 7

  Step 11
1 8 2
0 3 5
6 4 7
```

```
  Step 12
1 8 2
3 0 5
6 4 7

  Step 13
1 0 2
3 8 5
6 4 7

  Step 14
1 2 0
3 8 5
6 4 7

  Step 15
1 2 5
3 8 0
6 4 7

  Step 16
1 2 5
3 0 8
6 4 7

  Step 17
1 2 5
3 4 8
6 0 7
```

```
  Step 18
1 2 5
3 4 8
6 7 0

  Step 19
1 2 5
3 4 0
6 7 8

  Step 20
1 2 0
3 4 5
6 7 8

  Step 21
1 0 2
3 4 5
6 7 8

  Step 22
0 1 2
3 4 5
6 7 8
```

```
  Search with heuristic 2.
1 8 0
5 7 2
6 4 3

  Step 0
1 8 2
5 7 0
6 4 3

  Step 1
1 8 2
5 0 7
6 4 3

  Step 2
1 8 2
0 5 7
6 4 3

  Step 3
1 8 2
6 5 7
0 4 3

  Step 4
1 8 2
6 5 7
4 0 3
```

```
  Step 5
1 8 2
6 5 7
4 3 0

  Step 6
1 8 2
6 5 0
4 3 7

  Step 7
1 8 2
6 0 5
4 3 7

  Step 8
1 8 2
6 3 5
4 0 7

  Step 9
1 8 2
6 3 5
0 4 7

  Step 10
1 8 2
0 3 5
6 4 7
```

```
  Step 11
1 8 2
3 0 5
6 4 7

  Step 12
1 0 2
3 8 5
6 4 7

  Step 13
1 2 0
3 8 5
6 4 7

  Step 14
1 2 5
3 8 0
6 4 7

  Step 15
1 2 5
3 0 8
6 4 7

  Step 16
1 2 5
3 4 8
6 0 7
```

```
  Step 17
1 2 5
3 4 8
6 7 0

  Step 18
1 2 5
3 4 0
6 7 8

  Step 19
1 2 0
3 4 5
6 7 8

  Step 20
1 0 2
3 4 5
6 7 8

  Step 21
0 1 2
3 4 5
6 7 8
```