

E2 15-Puzzle Problem (IDA*)

17341068 Yangfan Jiang

September 8, 2019

Contents

1	IDA* Algorithm	2
1.1	Description	2
1.2	Pseudocode	2
2	Tasks	3
3	Codes	3
4	Results	7

1 IDA* Algorithm

1.1 Description

Iterative deepening A* (IDA*) was first described by Richard Korf in 1985, which is a graph traversal and path search algorithm that can find the shortest path between a designated start node and any member of a set of goal nodes in a weighted graph.

It is a variant of **iterative deepening depth-first search** that borrows the idea to use a heuristic function to evaluate the remaining cost to get to the goal from the **A* search algorithm**.

Since it is a depth-first search algorithm, its memory usage is lower than in A*, but unlike ordinary iterative deepening search, it concentrates on exploring the most promising nodes and thus does not go to the same depth everywhere in the search tree.

Iterative-deepening-A* works as follows: at each iteration, perform a depth-first search, cutting off a branch when its total cost $f(n) = g(n) + h(n)$ exceeds a given threshold. This threshold starts at the estimate of the cost at the initial state, and increases for each iteration of the algorithm. At each iteration, the threshold used for the next iteration is the minimum cost of all values that exceeded the current threshold.

1.2 Pseudocode





```
path          current search path (acts like a stack)
node          current node (last node in current path)
g            the cost to reach current node
f            estimated cost of the cheapest path (root..node..goal)
h(node)      estimated cost of the cheapest path (node..goal)
cost(node, succ) step cost function
is_goal(node) goal test
successors(node) node expanding function, expand nodes ordered by g + h(node)
ida_star(root) return either NOT_FOUND or a pair with the best path and its cost

procedure ida_star(root)
  bound := h(root)
  path := [root]
  loop
    t := search(path, 0, bound)
    if t = FOUND then return (path, bound)
    if t = ∞ then return NOT_FOUND
    bound := t
  end loop
end procedure

function search(path, g, bound)
  node := path.last
  f := g + h(node)
  if f > bound then return f
  if is_goal(node) then return FOUND
  min := ∞
  for succ in successors(node) do
    if succ not in path then
      path.push(succ)
      t := search(path, g + cost(node, succ), bound)
      if t = FOUND then return FOUND
      if t < min then min := t
    path.pop()
  end if
end for
return min
end function
```

2 Tasks

- Please solve 15-Puzzle problem by using IDA* (Python or C++). You can use one of the two commonly used heuristic functions: h_1 = the number of misplaced tiles. h_2 = the sum of the distances of the tiles from their goal positions.
- Here are 4 test cases for you to verify your algorithm correctness. You can also play this game (15puzzle.zip) for more information.

	<p>TextOut of Result</p> <pre> 11 3 1 7 4 6 8 2 15 9 10 13 14 12 5 0 LowerBound 36 moves A optimal solution 56 moves Used time 3 sec 13 10 8 6 9 12 5 13 10 8 12 15 14 5 13 12 15 14 5 13 14 9 4 11 3 1 6 4 11 3 1 6 4 2 8 10 12 15 10 8 7 4 2 11 3 5 9 10 11 3 6 2 3 7 8 12 </pre>		<p>TextOut of Result</p> <pre> 14 10 6 0 4 9 1 8 2 3 5 11 12 13 7 15 LowerBound 37 moves A optimal solution 49 moves Used time 0 sec 6 10 9 4 14 9 4 1 10 4 1 3 2 14 9 1 3 2 5 11 8 6 4 3 2 5 13 12 14 13 12 7 11 12 7 14 13 9 5 10 6 8 12 7 10 6 7 11 15 </pre>
	<p>TextOut of Result</p> <pre> 0 5 15 14 7 9 6 13 1 2 12 10 8 11 4 3 LowerBound 44 moves A optimal solution 62 moves Used time 4 sec 7 9 2 1 9 2 5 7 2 5 1 11 8 9 5 1 6 12 10 3 4 8 11 10 12 13 3 4 8 12 13 15 14 3 4 8 12 13 15 14 7 2 1 5 10 11 13 15 14 7 3 4 8 12 15 14 11 10 9 13 14 15 </pre>		<p>TextOut of Result</p> <pre> 6 10 3 15 14 8 7 11 5 1 0 2 13 12 9 4 LowerBound 32 moves A optimal solution 48 moves Used time 0 sec 9 12 13 5 1 9 7 11 2 4 12 13 9 7 11 2 15 3 2 15 4 11 15 8 14 1 5 9 13 15 7 14 10 6 1 5 9 13 14 10 6 2 3 4 8 7 11 12 </pre>

- Please send E02_YourNumber.pdf to ai_201901@foxmail.com, you can certainly use E02_15puzzle.tex as the L^AT_EX template.

3 Codes

```

1 # AI Experiment #2: 15 puzzle problem
2 # 2019/9/8
3 __author__ = 'Yangfan Jiang (jiangyf29@mail2.sysu.edu.cn)'
4
5 '''
6 Solve 15 puzzle problem by Iterative Deepening A*
7 '''
8
9 from copy import *
10 import numpy as np
11
12 # 2 D list: 4*4
13 puzzle = []
14
15
16 def h(puzzle):
17     '''

```

```

18     The sum of the distances of the tiles from their goal positions
19
20     Input:
21     puzzle: 2 D list stand for current state of puzzle
22
23     Returns:
24     sum(int): the value of h function
25     '''
26     sum = 0
27     for i in range(0, 4):
28         for j in range(0, 4):
29             if puzzle[i][j] != i*4+j+1 and puzzle[i][j] != 0:
30                 x = (puzzle[i][j]-1) // 4
31                 y = (puzzle[i][j]+3)%4
32                 sum += abs(x-i) + abs(y-j)
33     return sum
34
35
36 def is_goal(puzzle):
37     '''
38     Judge whether the state is finish
39
40     Input:
41     puzzle: 2 D list stand for current state of puzzle
42
43     Returns:
44     Boolean value
45     '''
46     for i in range(0, 4):
47         for j in range(0, 4):
48             if puzzle[i][j] != i*4+j+1 and i*4+j+1!=16:
49                 return False
50     return True
51
52
53 def successor(node):
54     '''
55     Find the successor of current state
56
57     Input:
58     node(2D list)
59
60     Returns:
61     next_states(list): neighbors of zero element
62     '''
63     index = 0
64     curr_num = node[0][0]
65     i = j = 0
66     while curr_num != 0:

```

```

67         index += 1
68         i = index//4
69         j = index%4
70         curr_num = node[i][j]
71     next_states = []
72     next_pos = [(i-1,j), (i,j+1), (i+1,j), (i,j-1)]
73     for pos in next_pos:
74         if 0<=pos[0]<=3 and 0<=pos[1]<=3:
75             next = deepcopy(node)
76             next[i][j] = next[pos[0]][pos[1]]
77             next[pos[0]][pos[1]] = 0
78             next_states.append(deepcopy(next))
79     return next_states
80
81 def cost(node, succ):
82     '''
83     cost for search 1 step
84     '''
85     return 1
86
87 def is_inverse(root):
88     '''
89     Judge whether the problem has a solution
90     if the #inverse sequence pair of original state is odd
91     there are definitely no solution
92
93     Input:
94     root(2D list): original state
95
96     Output:
97     0 or 1, represent for whether there is a solution
98     '''
99     num_inverse = 0
100     sequence = [i for item in root for i in item]
101     l = len(sequence)
102     for i in range(1, l):
103         for j in range(0, i):
104             if sequence[j] > sequence[i]:
105                 num_inverse += 1
106     return (num_inverse%2) == 0
107
108 def ida_star(root):
109     '''
110     Implementtation of Iterative Deepening A*
111     '''
112     if is_inverse(root):
113         print("No solution")
114         return '', '', ''
115

```

```

116     bound = h(root)
117     path = [root]
118     while True:
119         t = search(path, 0, bound)
120         if t == 'Found':
121             return t, bound, path
122         elif t == 'inf':
123             return 'Not Found'
124         bound = t
125         print(t)
126
127 def search(path, g, bound):
128     '''
129     Deep First Search
130     '''
131     node = path[-1]
132     f = g + h(node)
133     if f > bound:
134         return f
135     if is_goal(node):
136         return 'Found'
137
138     min = 1000000
139     for succ in successor(node):
140         if succ not in path:
141             path.append(succ)
142             t = search(path, g + cost(node, succ), bound)
143             if t == 'Found':
144                 return 'Found'
145             if t < min:
146                 min = t
147             path[:] = path[: -1]
148
149     return min
150
151 def get_path(path):
152     '''
153     Transform sequence of states into sequence of steps
154     each step is represented by a number that needs to be moved
155     '''
156     trace = []
157     l = len(path)
158     for i in range(0, l-1):
159         x = np.array(path[i])
160         y = np.array(path[i+1])
161         res = y - x
162         pos = np.where(res > 0.1)
163         pos_x = pos[0][0]
164         pos_y = pos[1][0]

```

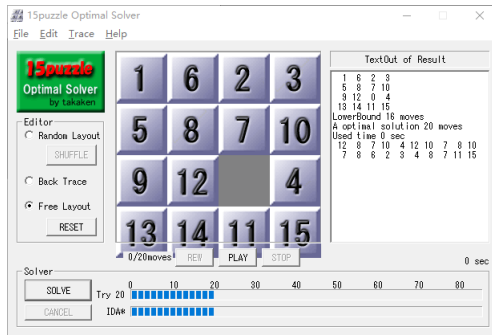
```

165         trace.append(y[pos_x][pos_y])
166     return trace
167
168 if __name__ == '__main__':
169     t = [[1,6,2,3],[5,8,7,10],[9,12,0,4],[13,14,11,15]]
170     x, y, z = ida_star(t)
171     trace = get_path(z)
172     print('----- solution -----')
173     print(trace)

```

4 Results

- The test results are shown in the figure below. The two samples require 20 and 48 steps, respectively
- For more details on the code and source code files, please visit <https://github.com/Yangfan-Jiang/Artificial-Intelligence-Fall-2019>



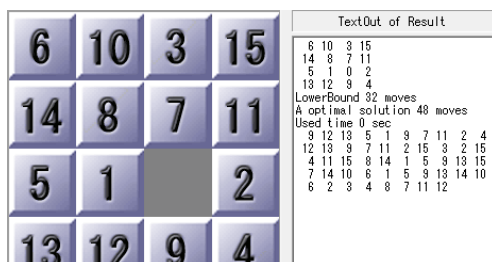
(a) Test Case1

```

D:\study\Artificial-Intelligence-Fall-2019\EXP\E02_20190904_15puzzle
16
18
20
----- solution -----
[12, 8, 7, 10, 4, 12, 10, 7, 8, 10, 7, 8, 6, 2, 3, 4, 8, 7, 11, 15]

```

(b) Result of Case1



(c) Test Case2

```

jyfar@amax:~/study$ python3 02.py
34
36
38
40
42
44
46
48
----- solution -----
[9, 12, 13, 5, 1, 9, 7, 11, 2, 4, 12, 13, 9, 7, 11, 2,
15, 3, 2, 15, 4, 11, 15, 8, 14, 1, 5, 9, 13, 15, 7, 14,
10, 6, 1, 5, 9, 13, 14, 10, 6, 2, 3, 4, 8, 7, 11, 12]

```

(d) Result of Case2

Figure 1: Results