School of Data and Computer Science, Sun Yat-sen University

E2 15-Puzzle Problem (IDA*)

18308045 Zhengyang Gu

September 11, 2020

Contents

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

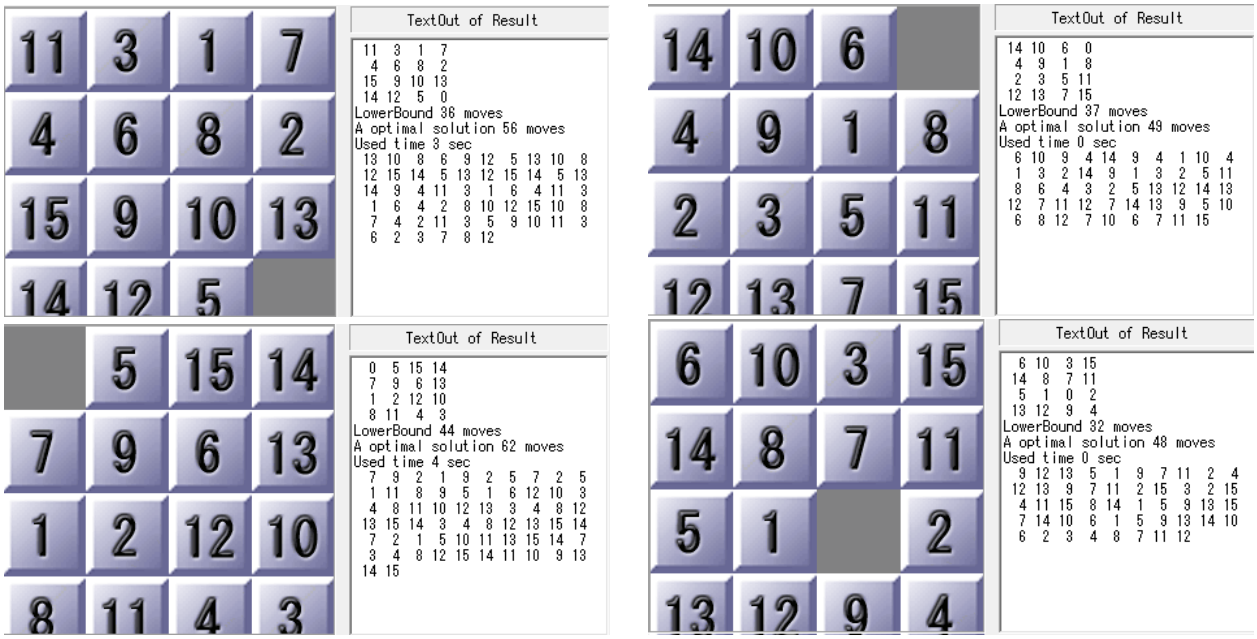
```
path              current search path (acts like a stack)
node              current node (last node in current path)
g                 the cost to reach current node
f                 estimated cost of the cheapest path (root..node..goal)
h(node)           estimated cost of the cheapest path (node..goal)
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: h1 = the number of misplaced tiles. h2 = 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.exe`) for more information.



- Please send `E02_YourNumber.pdf` to `ai_2020@foxmail.com`, you can certainly use `E02_15puzzle.tex` as the LaTeXtemplate.

3 Codes

```python
import time
path = list()
# 通过值查目标位置
goal_reverse = ((3, 3), (0, 0), (0, 1), (0, 2),
                (0, 3), (1, 0), (1, 1), (1, 2),
                (1, 3), (2, 0), (2, 1), (2, 2),
                (2, 3), (3, 0), (3, 1), (3, 2))
row = 4
col = 4
range_row = range(row)
range_col = range(col)
puzzle = list()
# 曼哈顿距离，只调用一次，后续动态修改f值
def h():
    ans = 0
    for i in range_row:
        for j in range_col:
```

```
            if puzzle[i][j] is 0:
                continue
            goal_pos = goal_reverse[puzzle[i][j]]
            ans += abs(i - goal_pos[0]) + abs(j - goal_pos[1])
    return ans
cost = 1
# 通过位置查目标值
goal = ((1, 2, 3, 4),
        (5, 6, 7, 8),
        (9, 10, 11, 12),
        (13, 14, 15, 0))
def is_goal():
    for i in range_row:
        for j in range_col:
            if puzzle[i][j] is 0:
                continue
            if puzzle[i][j] is not goal[i][j]:
                return False
    return True
# 记录0所在的位置，方便更新
zero_pos
directions = ((-1, 0), (1, 0), (0, -1), (0, 1))
def successors():
    global zero_pos
    ans = list()
    for direction in directions:
        succ = (zero_pos,
                (zero_pos[0] + direction[0],
                zero_pos[1] + direction[1]))
        if 0 <= succ[1][0]\
                and succ[1][0] < row\
                and 0 <= succ[1][1]\
                and succ[1][1] < col:
            ans.append(succ)
    return ans
# 当前图的hash值，可以放入set中用于路经检测
# list类型的puzzle无法放入set中
cur_hash = 0
# 初始化cur_hash，后续只对变化元素修改cur_hash
def hash():
    global cur_hash
    base = 1
    for i in range_row:
        for j in range_col:
            cur_hash += base * puzzle[i][j]
            base *= 16
inf = float("inf")
vis = set()
def ida_star():
```

```
67      global zero_pos
68      stt = h()
69      bound = stt
70      vis.add(hash())
71      for i in range_row:
72          for j in range_col:
73              if puzzle[i][j] is 0:
74                  zero_pos = (i, j)
75                  break
76          else:
77              continue
78          break
79      while 1:
80          t = search(path, stt, bound)
81          if t is True:
82              return (path, bound)
83          if t == inf:
84              return False
85          bound = t
86  def search(path, f, bound):
87      global zero_pos
88      global cur_hash
89      if f > bound:
90          return f
91      if is_goal():
92          return True
93      min = inf
94      for succ in successors():
95          succ_value = puzzle[succ[1][0]][succ[1][1]]
96          last_hash = cur_hash
97          cur_hash += succ_value
98          * (16 ** (succ[0][0] * col + succ[0][1])
99          - 16 ** (succ[1][0] * col + succ[1][1]))
100         puzzle[succ[0][0]][succ[0][1]] = succ_value
101         puzzle[succ[1][0]][succ[1][1]] = 0
102         zero_pos = succ[1]
103         if cur_hash not in vis:
104             path.append(succ)
105             vis.add(cur_hash)
106             goal_pos = goal_reverse[succ_value]
107             t = search(path,
108                        f + abs(succ[0][0] - goal_pos[0]) +
109                        abs(succ[0][1] - goal_pos[1])
110                        - (abs(succ[1][0] - goal_pos[0]) +
111                            abs(succ[1][1] - goal_pos[1])) + cost,
112                        bound)
113             if t is True:
114                 return True
115             if t < min:
```

```
                min = t
            vis.remove(cur_hash)
            path.pop()
        zero_pos = succ[0]
        puzzle[succ[1][0]][succ[1][1]] = succ_value
        puzzle[succ[0][0]][succ[0][1]] = 0
        cur_hash = last_hash
    return min
def read_15puzzle(file_name):
    with open(file_name) as f:
        for i in range_row:
            line = f.readline()
            if not line:
                break
            puzzle.append([int(i) for i in line.split()])
read_15puzzle("test4.txt")
for i in puzzle:
    for j in i:
        print(j, end = '\t')
    print()
start = time.perf_counter()
ans = ida_star()
end = time.perf_counter()
print("Time used:", end - start)
print(ans)
```

4 Results

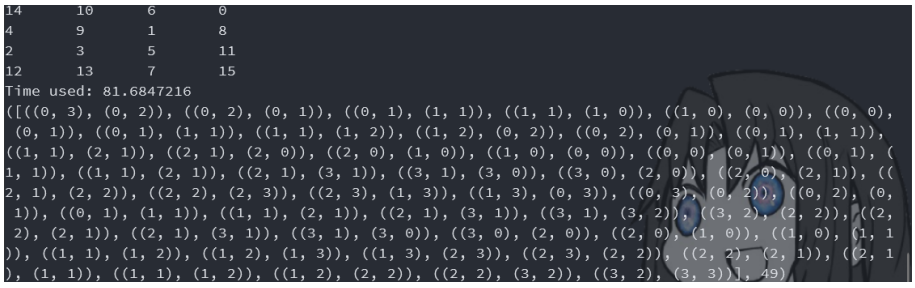


Figure 1: result2

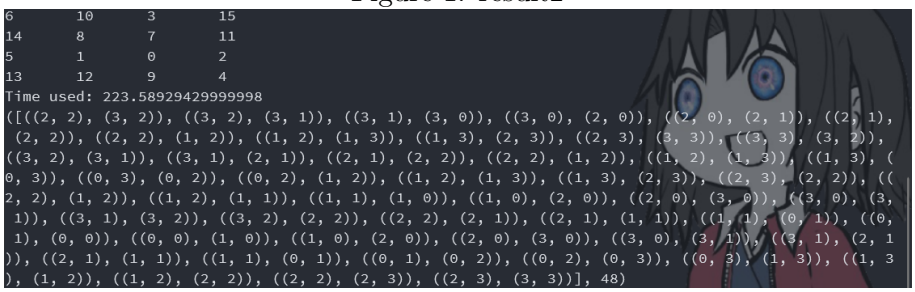


Figure 2: result4