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

E2 15-Puzzle Problem (IDA*)

17341068 Yangfan Jiang

September 8, 2019

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.

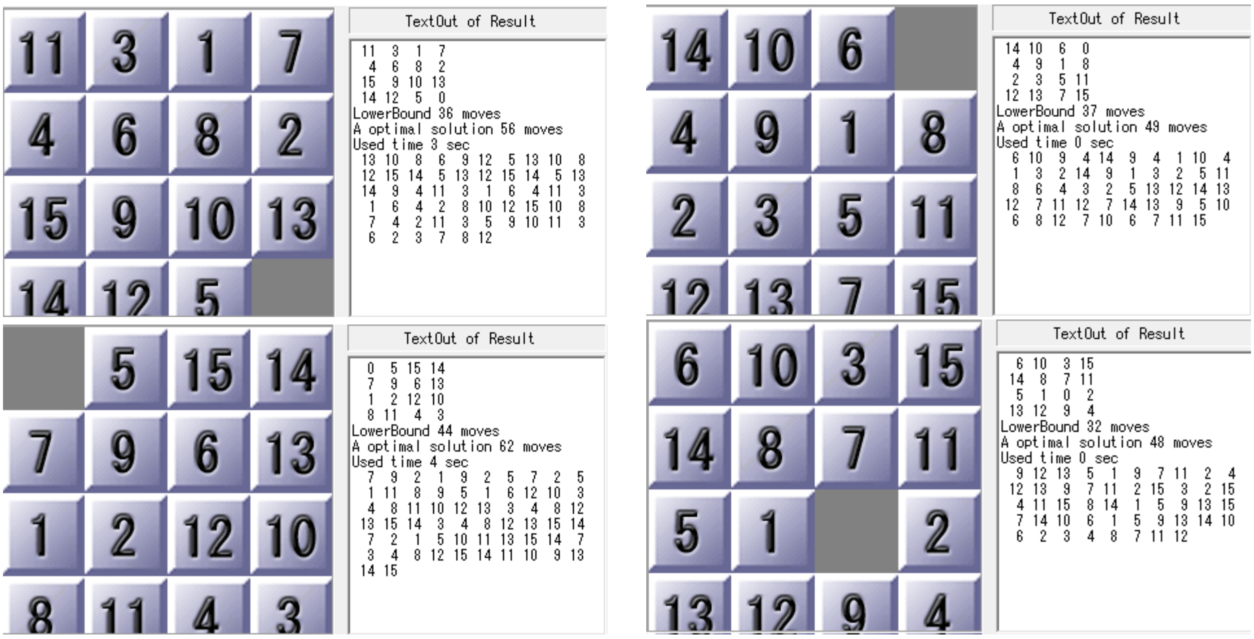
Iterative-deepening-A* works as follows: at each iteration, perform a depth-first search, cutting off a branch when its total cost $f(n) = g(n) + h(n)$ exceeds a given threshold. This threshold starts at the estimate of the cost at the initial state, and increases for each iteration of the algorithm. 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.zip`) for more information.



- Please send `E02_YourNumber.pdf` to `ai_201901@foxmail.com`, you can certainly use `E02_15puzzle.tex` as the LaTeXtemplate.

3 Codes

```
# AI Experiment #2: 15 puzzle problem
# 2019/9/8
__author__ = 'Yangfan Jiang (jiangyf29@mail2.sysu.edu.cn)'

'''
Solve 15 puzzle problem by Iterative Deepening A*
'''

from copy import *
import numpy as np

# 2 D list: 4*4
puzzle = []


def h(puzzle):
    '''
```

```
    The sum of the distances of the tiles from their goal positions

    Input:
    puzzle: 2 D list stand for current state of puzzle

    Returns:
    sum(int): the value of h function
    '''
    sum = 0
    for i in range(0, 4):
        for j in range(0, 4):
            if puzzle[i][j] != i*4+j+1 and puzzle[i][j] != 0:
                x = (puzzle[i][j]-1) // 4
                y = (puzzle[i][j]+3)%4
                sum += abs(x-i) + abs(y-j)
    return sum


def is_goal(puzzle):
    '''
    Judge whether the state is finish

    Input:
    puzzle: 2 D list stand for current state of puzzle

    Returns:
    Boolean value
    '''
    for i in range(0, 4):
        for j in range(0, 4):
            if puzzle[i][j] != i*4+j+1 and i*4+j+1!=16:
                return False
    return True


def successor(node):
    '''
    Find the successor of current state

    Input:
    node(2D list)

    Returns:
    next_states(list): neighbors of zero element
    '''
    index = 0
    curr_num = node[0][0]
    i = j = 0
    while curr_num != 0:
```

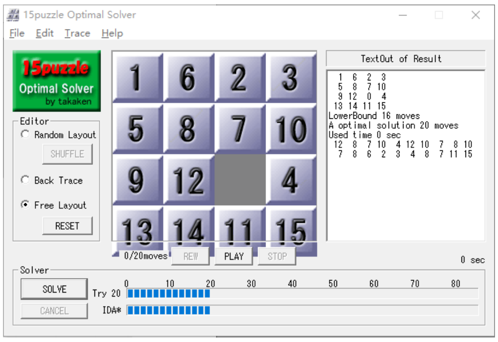
```
        index += 1
        i = index//4
        j = index%4
        curr_num = node[i][j]
    next_states = []
    next_pos = [(i-1,j), (i,j+1), (i+1,j), (i,j-1)]
    for pos in next_pos:
        if 0<=pos[0]<=3 and 0<=pos[1]<=3:
            next = deepcopy(node)
            next[i][j] = next[pos[0]][pos[1]]
            next[pos[0]][pos[1]] = 0
            next_states.append(deepcopy(next))
    return next_states

def cost(node, succ):
    '''
    cost for search 1 step
    '''
    return 1

def is_inverse(root):
    '''
    Judeg whether the problem has a solution
    if the #inverse sequence pair of original state is odd
    there are definitely no solution

    Input:
    root(2D list): original state

    Output:
    0 or 1, repersent for whether there is a solution
    '''
    num_inverse = 0
    sequence = [i for item in root for i in item]
    l = len(sequence)
    for i in range(1, l):
        for j in range(0, i):
            if sequence[j] > sequence[i]:
                num_inverse += 1
    return (num_inverse%2) == 0

def ida_star(root):
    '''
    Implementtation of Iterative Deepening A*
    '''
    if is_inverse(root):
        print("No solution")
        return '','',''

```

```
    bound = h(root)
    path = [root]
    while True:
        t = search(path, 0, bound)
        if t == 'Found':
            return t, bound, path
        elif t == 'inf':
            return 'Not Found'
        bound = t
        print(t)

def search(path, g, bound):
    '''
    Deep First Search
    '''
    node = path[-1]
    f = g + h(node)
    if f > bound:
        return f
    if is_goal(node):
        return 'Found'

    min = 1000000
    for succ in successor(node):
        if succ not in path:
            path.append(succ)
            t = search(path, g + cost(node, succ), bound)
            if t == 'Found':
                return 'Found'
            if t < min:
                min = t
            path[:] = path[:-1]

    return min

def get_path(path):
    '''
    Transform sequence of states into sequence of steps
    each step is repersented by a number that needs to be moved
    '''
    trace = []
    l = len(path)
    for i in range(0, l-1):
        x = np.array(path[i])
        y = np.array(path[i+1])
        res = y - x
        pos = np.where(res > 0.1)
        pos_x = pos[0][0]
        pos_y = pos[1][0]
```

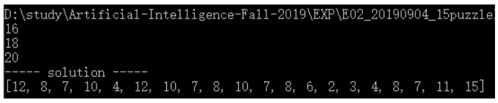
```
        trace.append(y[pos_x][pos_y])
    return trace

if __name__ == '__main__':
    t = [[1,6,2,3],[5,8,7,10],[9,12,0,4],[13,14,11,15]]
    x, y, z = ida_star(t)
    trace = get_path(z)
    print('------ solution ------')
    print(trace)
```

4 Results

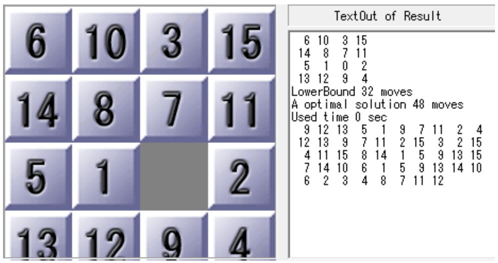
- The test results are shown in the figure below. The two samples require 20 and 48 steps, respectively
- For more details on the code and source code files, please visit https://github.com/Yangfan-Jiang/Artificial-Intelligence-Fall-2019



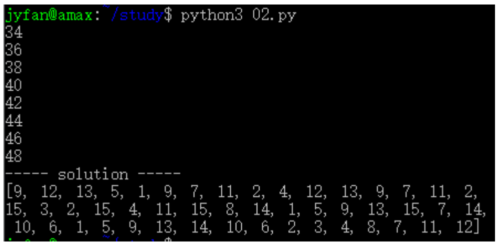
(a) Test Case1



(b) Result of Case1



(c) Test Case2



(d) Result of Case2

Figure 1: Results