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

E2 15-Puzzle Problem (IDA*)

17341175 xuzhicheng

September 5, 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.

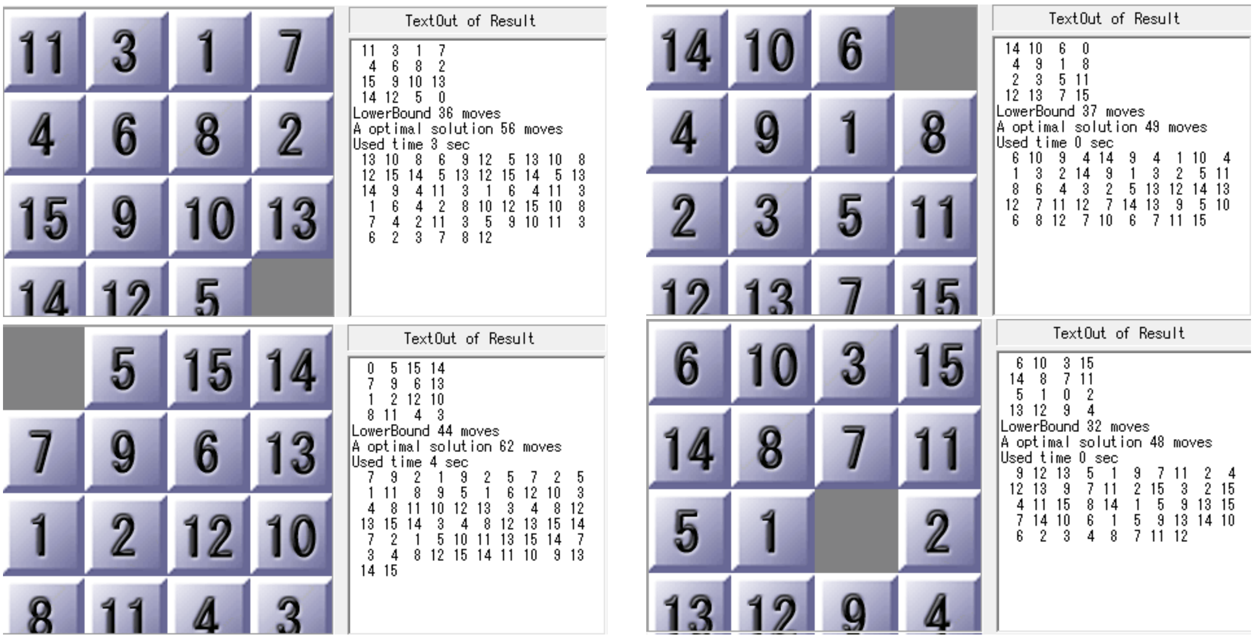
1.2 Pseudocode

```
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

```
goal = {}
index = 1
for row in range(4):
    for col in range(4):
        goal[index] = (row, col)
        index += 1
del goal[16]
goal[0] = (3, 3)
print(goal)

def h(node):
    sum = 0
    for row in range(4):
        for col in range(4):
            a = node[row][col]
            (x, y) = goal[a]
            sum += abs(x-row)+abs(y-col)
```

```python
    return sum


def is_goal(node):
    index = 1
    for row in node:
        for col in row:
            if (index != col):
                break
            index += 1
    return index == 16

move = [(1, 0), (-1, 0), (0, 1), (0, -1)]


def successors(node):
    x, y = 0, 0
    for row in range(4):
        for col in range(4):
            if node[row][col] == 0:
                x, y =row, col
    success = []
    for i,j in move:
        r,l = x+i, y+j
        if r<4 and r>-1 and l>-1 and l<4:
            temp = [[num for num in row]for row in node]
            temp[x][y] = temp[r][l]
            temp[r][l] = 0
            success.append(temp)
    return sorted(success, key=lambda x:h(x))


def search(path, g, bound):
    node = path[-1]
    f = g + h(node)

    if f > bound:
        return f
    if is_goal(node):
        return -1
    min = float("inf")
    for su in successors(node):
        if su not in path:
            path.append(su)
            ff = search(path, g+1, bound)
            if ff == -1:
                return -1
            if ff < min:
                min = ff
```

```
            path.pop()
    return min


def ida_star(root):
    bound = h(root)
    path = [root]
    while 1:
        print("lowerbound", bound)
        t = search(path, 0, bound)

        if t == -1:
            return path, bound
        if t > 60:
            return [], bound
        bound = t




root = [[2, 7, 3, 4], [1, 0, 12, 8], [5, 6, 9, 10], [13, 14, 11, 15]]
# root=[[0, 5, 15, 14], [7, 9, 6, 13], [1, 2, 12, 10], [8, 11, 4, 3]]
#root = [[11, 3, 1, 7], [4, 6, 8, 2], [15, 9, 10, 13], [14, 12, 5, 0]]
(path, bound) = ida_star(root)
procedure = 0
for p in path:
    print("procedure", procedure)
    procedure += 1
    for row in p:
        print(row)

print("optimal solution:", len(path)-1, "moves")
print("lowerbound: ", bound)
```

4 Results

```
procedure 18
[1, 2, 3, 4]
[5, 6, 7, 8]
[9, 10, 0, 12]
[13, 14, 11, 15]
procedure 19
[1, 2, 3, 4]
[5, 6, 7, 8]
[9, 10, 11, 12]
[13, 14, 0, 15]
procedure 20
[1, 2, 3, 4]
[5, 6, 7, 8]
[9, 10, 11, 12]
[13, 14, 15, 0]
optimal solution: 20 moves
lowerbound:  26

Process finished with exit code 0
```