人工智能基础实验报告

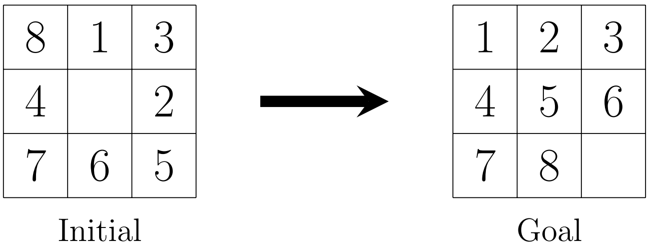
刘砺志

（2014 级计算机 1 班　22920142203873）

本文是人工智能基础——A* 算法解决八数码难题的实验报告。

1　实验概述

本次实验要求实现 A* 算法求解八数码问题（Eight Puzzle Problem）。所谓八数码问题，就是在 3×3 的方格棋盘上，摆放着 1 到 8 这八个数码，有 1 个方格是空的，如图 1所示，要求对空格执行空格左移、空格右移、空格上移和空格下移这四个操作使得棋盘从初始状态到目标状态。



8	1	3
4		2
7	6	5

1	2	3
4	5	6
7	8	

图 1　八数码问题

2　实验原理

2.1　A* 搜索算法（A* search algorithm）

下面的伪代码描述了 A* 算法的流程：

```
A*($start, goal$)
// The set of nodes already evaluated
closedSet := {}
```

```
// The set of currently discovered nodes that are not evaluated yet.
// Initially, only the start node is known.
openSet := {start}

// For each node, which node it can most efficiently be reached from.
// If a node can be reached from many nodes, cameFrom will eventually contain the
// most efficient previous step.
cameFrom := an empty map

// For each node, the cost of getting from the start node to that node.
gScore := map with DEFAULT value of Infinity

// The cost of going from start to start is zero.
gScore[start] := 0

// For each node, the total cost of getting from the start node to the goal
// by passing by that node. That value is partly known, partly heuristic.
fScore := map with DEFAULT value of Infinity

// For the first node, that value is completely heuristic.
fScore[start] := heuristic_cost_estimate(start, goal)

while openSet is not empty
    current := the node in openSet having the lowest fScore[] value
    if current = goal
        return reconstruct_path(cameFrom, current)

    openSet.Remove(current)
    closedSet.Add(current)

    for each neighbor of current
        if neighbor in closedSet
            continue // Ignore the neighbor which is already evaluated.
        if neighbor not in openSet          // Discover a new node
            openSet.Add(neighbor)

        // The distance from start to a neighbor
        tentative_gScore := gScore[current] + dist_between(current, neighbor)
        if tentative_gScore >= gScore[neighbor]
            continue                        // This is not a better path.

        // This path is the best until now. Record it!
        cameFrom[neighbor] := current
        gScore[neighbor] := tentative_gScore
        fScore[neighbor] := gScore[neighbor] + heuristic_cost_estimate(neighbor, goal)
return FAILURE
```

```
Reconstruct-Path($cameFrom, current$)
total_path := [current]
while current in cameFrom.Keys
    current := cameFrom[current]
    total_path.append(current)
return total_path
```

2.2 最好优先搜索（Best-first search）

贪婪最好优先搜索的伪代码如下：

```
GreedyBFS
insert (state = initial_state, h = initial_heuristic, counter = 0) into search_queue;

while search_queue not empty do
    current_queue_entry = pop item from front of search_queue;
    current_state = state from current_queue_entry;
    current_heuristic = heuristic from current_queue_entry;
    starting_counter = counter from current_queue_entry;
    aplicable_actions = array of actions applicable in current_state;
    for all index ?i in applicable_actions $\leq$ starting_counter do
        current_action = applicable_actions[?i];
        successor_state = current_state.apply(current_action);

        if successor_state is goal then
            return plan and exit;
        successor_heuristic = heuristic value of successor_state;

        if successor_heuristic $<$ current_heuristic then
            insert (current_state, current_heuristic, ?i+1) at front of search_queue;
            insert (successor_state, successor_heuristic, 0) at front of search_queue;
            break for;
        else
            insert (successor_state, successor_heuristic, 0) into search_queue;

exit - no plan found;
```

2.3 迭代加深 A* 搜索算法（Iterative deepening A*）

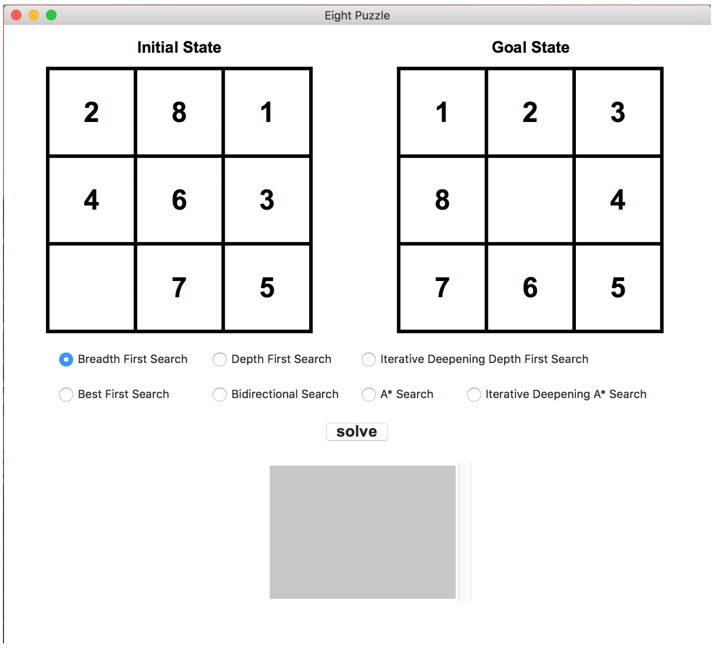
迭代加深 A* 搜索算法的伪代码如下:

```
IDA-STAR(root)
bound := h(root)
path := [root]
loop
    t := search(path, 0, bound)
    if t = FOUND then return (path, bound)
    if t = $\infty$ then return NOT_FOUND
    bound := t
end loop
```

```
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 := $\infty$
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
```

3 实验结果

我使用 Python 3.6.0 编写了所有程序，并且利用 Tkinter 库绘制了图形界面便于操作，用户界面如图 2所示。



Eight Puzzle

Initial State

2	8	1
4	6	3
	7	5

Goal State

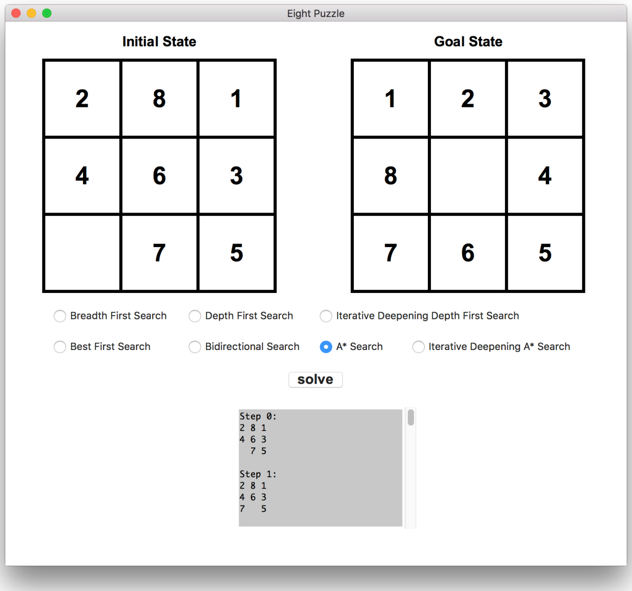
1	2	3
8		4
7	6	5

● Breadth First Search ○ Depth First Search ○ Iterative Deepening Depth First Search

○ Best First Search ○ Bidirectional Search ○ A* Search ○ Iterative Deepening A* Search

图 2 用户界面效果

使用 A* 算法的运行结果如图 3所示。



Initial State

2	8	1
4	6	3
	7	5

Goal State

1	2	3
8		4
7	6	5

○ Breadth First Search ○ Depth First Search ○ Iterative Deepening Depth First Search

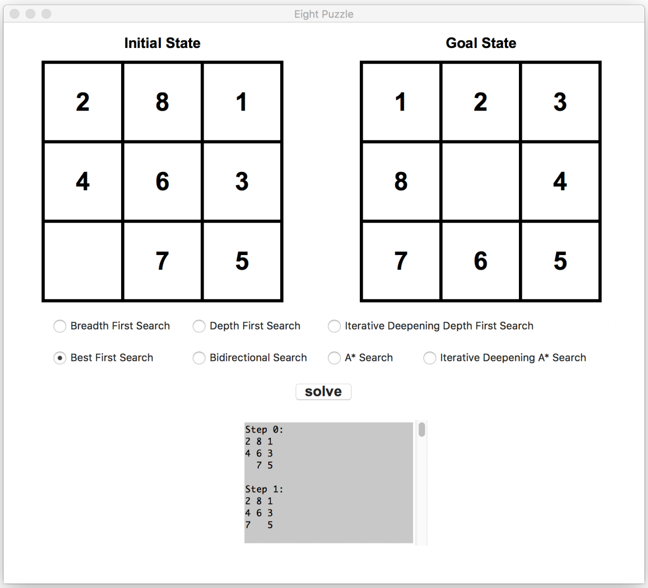
○ Best First Search ○ Bidirectional Search ● A* Search ○ Iterative Deepening A* Search

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

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

图 3　使用 A* 算法求解八数码问题

使用最好优先搜索算法的运行结果如图 4所示。



Initial State

2	8	1
4	6	3
	7	5

Goal State

1	2	3
8		4
7	6	5

○ Breadth First Search ○ Depth First Search ○ Iterative Deepening Depth First Search

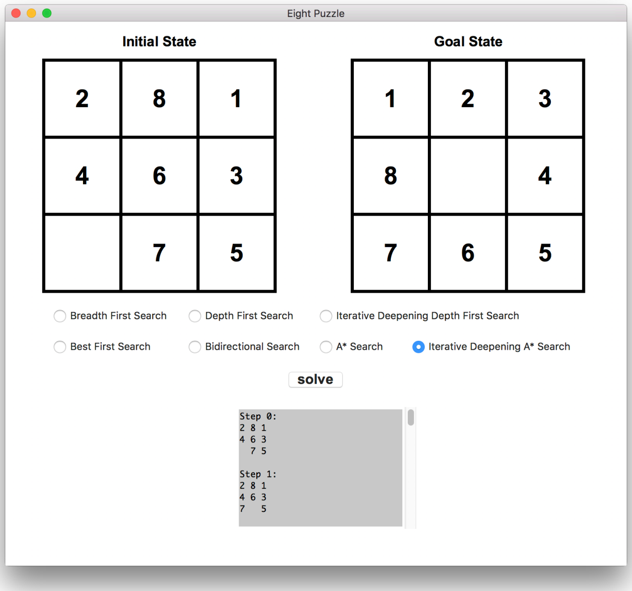
● Best First Search ○ Bidirectional Search ○ A* Search ○ Iterative Deepening A* Search

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

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

图 4　使用最好优先搜索算法求解八数码问题

使用 IDA* 算法的运行结果如图 5所示。



Initial State

2	8	1
4	6	3
	7	5

Goal State

1	2	3
8		4
7	6	5

○ Breadth First Search ○ Depth First Search ○ Iterative Deepening Depth First Search

○ Best First Search ○ Bidirectional Search ○ A* Search ● Iterative Deepening A* Search

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

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

图 5 使用 IDA* 算法求解八数码问题

参考文献

[1] 人工智能，Nils J. Nilsson 著，郑扣根，庄越挺译，潘云鹤校，北京：机械工业出版社，2000 年 9 月

[2] https://en.wikipedia.org/wiki/A*_search_algorithm

[3] http://www.cs.cmu.edu/afs/cs/project/jair/pub/volume28/coles07a-html/node11.html#modifiedbestfs

[4] https://en.wikipedia.org/wiki/Iterative_deepening_A*