中山大学计算机学院

人工智能

本科生实验报告

（2022学年春季学期）

课程名称：Artificial Intelligence

教学班级	信计系统结构方向	专业（方向）	ICS
学号	20337268	姓名	张文沁

一、 实验题目

启发式搜索算法解决15数码问题

二、 实验内容

1. 算法原理

$$f(n) = g(n) + h(h)$$

`其中 f(n) 是从初始点经由节点n到目标点的估价函数，g(n) 是在状态空间中从初始节点到n节点的实际代价，h(n) 是从n到目标节点最佳路径的估计代价。h*(n)是从n到目标节点最佳路径的实际代价,那么整个个启发式搜索过程必须保证h(n)<=h*(n)，否则搜索出错。h(n)和h*(x)越接近，算法越优`

启发式函数采用曼哈顿函数：

曼哈顿距离：

定义曼哈顿距离的正式意义为L1-距离或城市区块距离，也就是在欧几里德空间的固定直角坐标系上两点所形成的线段对轴产生的投影的距离总和。例如在平面上，坐标（x1,y1）的点P1与坐标（x2, y2）的点P2的曼哈顿距离为：|x1 - x2| + |y1 - y2|。

2. 关键代码展示（带注释）

1. 初始化：

```
# 方向数组,同3-1实验，四个方向
dx = [0, -1, 0, 1]
dy = [1, 0, -1, 0]
OPEN = [] #open表，记录剩余点
CLOSE = set() # close表，用于判断是否重复
path = []
# 最终状态
end = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0)
'''
最开始设置为列表,但是运行时间太长，改变为元组
```

```
end = [
    [1,2,3,4],
    [5,6,7,8],
    [9,10,11,12],
    [13,14,15,0]
]
'''
# 初始状态测例集
init = [
     (1, 2, 4, 8, 5, 7, 11, 10, 13, 15, 0, 3, 14, 6, 9, 12),
     (5, 1, 3, 4, 2, 7, 8, 12, 9, 6, 11, 15, 0, 13, 10, 14),
     (14, 10, 6, 0, 4, 9, 1, 8, 2, 3, 5, 11, 12, 13, 7, 15),
     (6, 10, 3, 15, 14, 8, 7, 11, 5, 1, 0, 2, 13, 12, 9, 4)
    ]
```

2. 设置节点状态：

```
class Node(object):
    def __init__(self, gn=0, hn=0, state=None, parent=None):
        self.gn = gn #g(n)
        self.hn = hn #h(n)
        self.fn = self.gn + self.hn #f(n)
        self.state = state
        self.parent = parent

    def __less_than__(self, node):
        if self.fn == node.fn:
            return self.gn > node.gn
        return self.fn < node.fn
```

3. 曼哈顿函数：

```
def manhattan(state):
    M = 0
    for t in range(16):
        if state[t] == end[t] or state[t]== 0:
            continue
        else:
            x =  (state[t] - 1) // 4    # 最终坐标
            y =  state[t] - 4 * x - 1
            dx = t // 4 # 实际坐标
            dy = t % 4
            M += (abs(x - dx) + abs(y - dy))
    return M
```

4. 拓展close表：

```
def generateChild():  # 生成子结点
   movetable = [] # 针对数码矩阵上每一个可能的位置，生成其能够移动的方向列表
   for i in range(16):
       x, y = i % 4, i // 4
       moves = []
       if x > 0: moves.append(-1)    # 左移
       if x < 3: moves.append(+1)    # 右移
       if y > 0: moves.append(-4)    # 上移
       if y < 3: moves.append(+4)    # 下移
```

```
        movetable.append(moves)
    def children(state):
        idxz = state.index(0) # 寻找数码矩阵上0的坐标
        l = list(state) # 将元组转换成list，方便进行元素修改
        for m in movetable[idxz]:
            l[idxz] = l[idxz + m] # 数码交换位置
            l[idxz + m] = 0
            yield(1, tuple(l)) # 临时返回
            l[idxz + m] = l[idxz]
            l[idxz] = 0
    return children
```

5. 主要算法：

```
def A_star(start, Fx): #start为起始结点，Fx为启发式函数
    root = Node(0, 0, start, None) #gn, hn,state, parent
    OPEN.append(root)
    heapq.heapify(OPEN)
    CLOSE.add(start)
    while len(OPEN) != 0: #存在未拓展的节点
        top = heapq.heappop(OPEN)
        global node_num  # 扩展的结点数
        node_num += 1
        if top.state == end: # 目标检测
            return print_path(top) #对路径进行打印
        generator = generateChild() #生成子结点
        for cost, state in generator(top.state):
            if state in CLOSE: #这里进行环检测
                continue
            CLOSE.add(state)
            child = Node(top.gn+cost, Fx(state), state,top)
            heapq.heappush(OPEN, child) # 将child加入优先队列中
```

三、 实验结果及分析

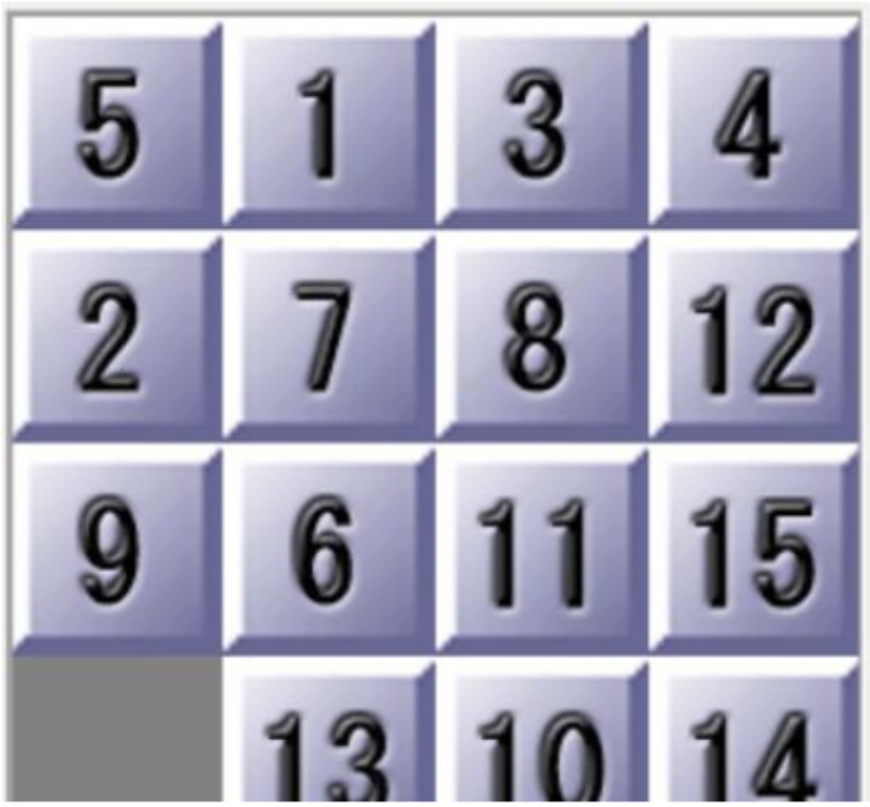
1. 实验结果展示示例

样例1

结果:

```
Step: 1
[ 1  2  4  8  5  7 11 10 13  0 15  3 14  6  9 12]
Step: 2
[ 1  2  4  8  5  7 11 10 13  6 15  3 14  0  9 12]
```

```
Step: 21
[ 1  2  3  4  5  6  7  8  9 10 11  0 13 14 15 12]
Step: 22
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15  0]
Test 1, Step 22
Time 0.012008
```

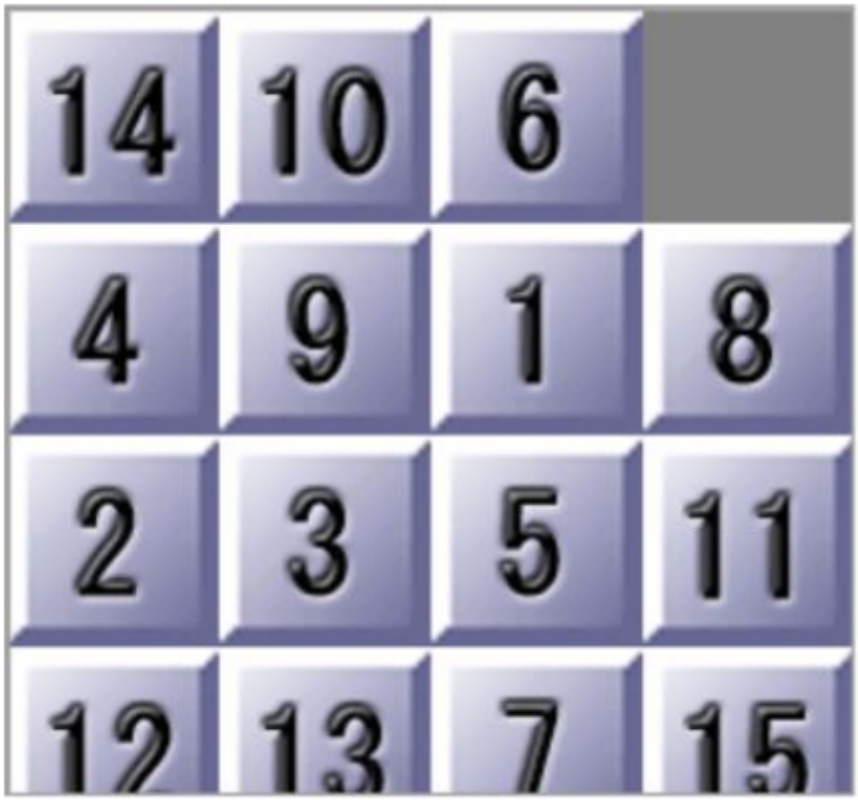


样例2

结果:

```
Step: 1
[ 5  1  3  4  2  7  8 12  9  6 11 15 13  0 10 14]
Step: 2
[ 5  1  3  4  2  7  8 12  9  6 11 15 13 10  0 14]
```

```
Step: 14
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14  0 15]
Step: 15
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15  0]
Test 2, Step 15
Time 0.001008
```

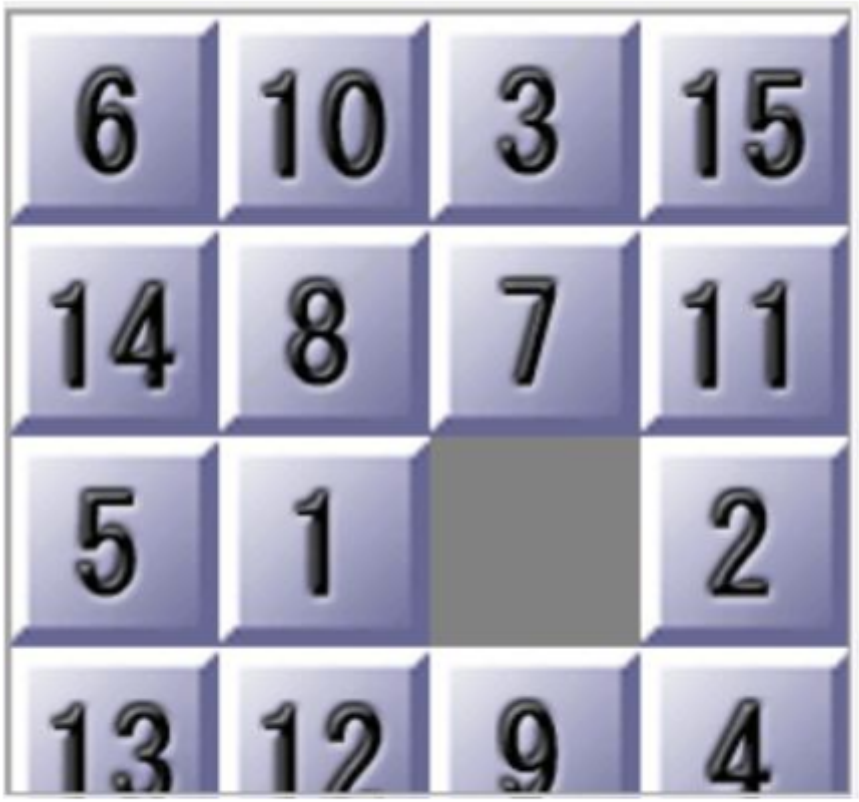


样例3

结果:

```
Step: 1
[14 10  0  6  4  9  1  8  2  3  5 11 12 13  7 15]
Step: 2
[14  0 10  6  4  9  1  8  2  3  5 11 12 13  7 15]
```

```
Step: 52
[ 1  2  3  4  5  6  7  8  9 10 11  0 13 14 15 12]
Step: 53
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15  0]
Test 3, Step 53
Time 508.035001
```



样例4

结果:

```
Step: 1
[ 6 10  3 15 14  8  7 11  5  1  9  2 13 12  0  4]
Step: 2
[ 6 10  3 15 14  8  7 11  5  1  9  2 13  0 12  4]
```

```
Step: 47
[ 1  2  3  4  5  6  7  8  9 10 11  0 13 14 15 12]
Step: 48
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15  0]
Test 4, Step 48
Time 189.644623
```