同济大学
TONGJI UNIVERSITY

人工智能课程设计实验报告

实 验 题 目：八数码实验

组　　员：苗成林 1851804

赵昊堃 1852877

唐骏龙 1851785

指 导 老 师：武　妍

目录

1 实验概述

1.1 实验目的

利用A*算法求解八数码问题，并在求解过程中进一步体会启发式搜索、估价函数以及A*算法的定义和内涵，在设计算法过程中熟悉A*算法的思想以及具体实现方式。

1.2 实验内容

• 用A*算法求解八数码问题，并基于相同的起始和目标状态，使用两种估价函数来设计算法，观察并对比两种估价函数对于算法效率的影响，包括扩展节点数、生成节点数以及运行时间等指标，对比后给出相应的性能分析。

• 在代码运行界面给出从起始状态到目标状态中间所经历的所有状态，并画出搜索过程所生成的搜索树。

2 实验方案设计

2.1 总体设计思路与总体架构

对于本次实验，我们小组考虑使用 Python 语言来进行实现，主要是基于以下考虑：

• 语法简单，实现相同功能时所需要的代码量相对较小

• 有许多便于使用的库，比如 matplotlib——Python中最著名的绘图库，可用于绘制寻找最佳路径的搜索树

• 编译速度相对更快

因此，基于以上考虑，我们决定使用Python语言实现A*算法来解决八数码问题，并且在输出最短路径的同时，一并绘制出对应的搜索树。

算法的总体思路如下：

首先，用户手动输入起始状态与目标状态所对应的序列（其中空白区域用0表示），程序先判断两个状态的逆序数奇偶性是否相同。若不一致，则直接输出“无解”，否则，开始进入搜索环节。

在正式搜索环节中，定义两个列表openlist和closelist，其中，openlist用于对现有结点进行排序并找出f(即估价函数值)最小的结点，而closelist则用于按照先后顺序存放所有遍历过的结点，便于后续搜索树的绘制。

在搜索的过程中，先判断openlist是否为空，若为空则输出“无解”，否则，取出openlist

的第一个元素，因为在A*算法函数末尾会对openlist的结点按照f值进行排序，因此此时取到的元素一定是当前f值最小的结点。若该结点状态与目标状态相同，则搜索完毕。否则，将其从openlist中取出，并找到空白格(即0)所在的位置，从而便于进行移动操作。将每一个移动后可能的状态当作新的结点，计算其估价函数的值，并将其同时加入到openlist与closelist中，最后再将openlist中元素按照f值进行排序。此时，一轮循环结束。

算法的具体流程图如下：

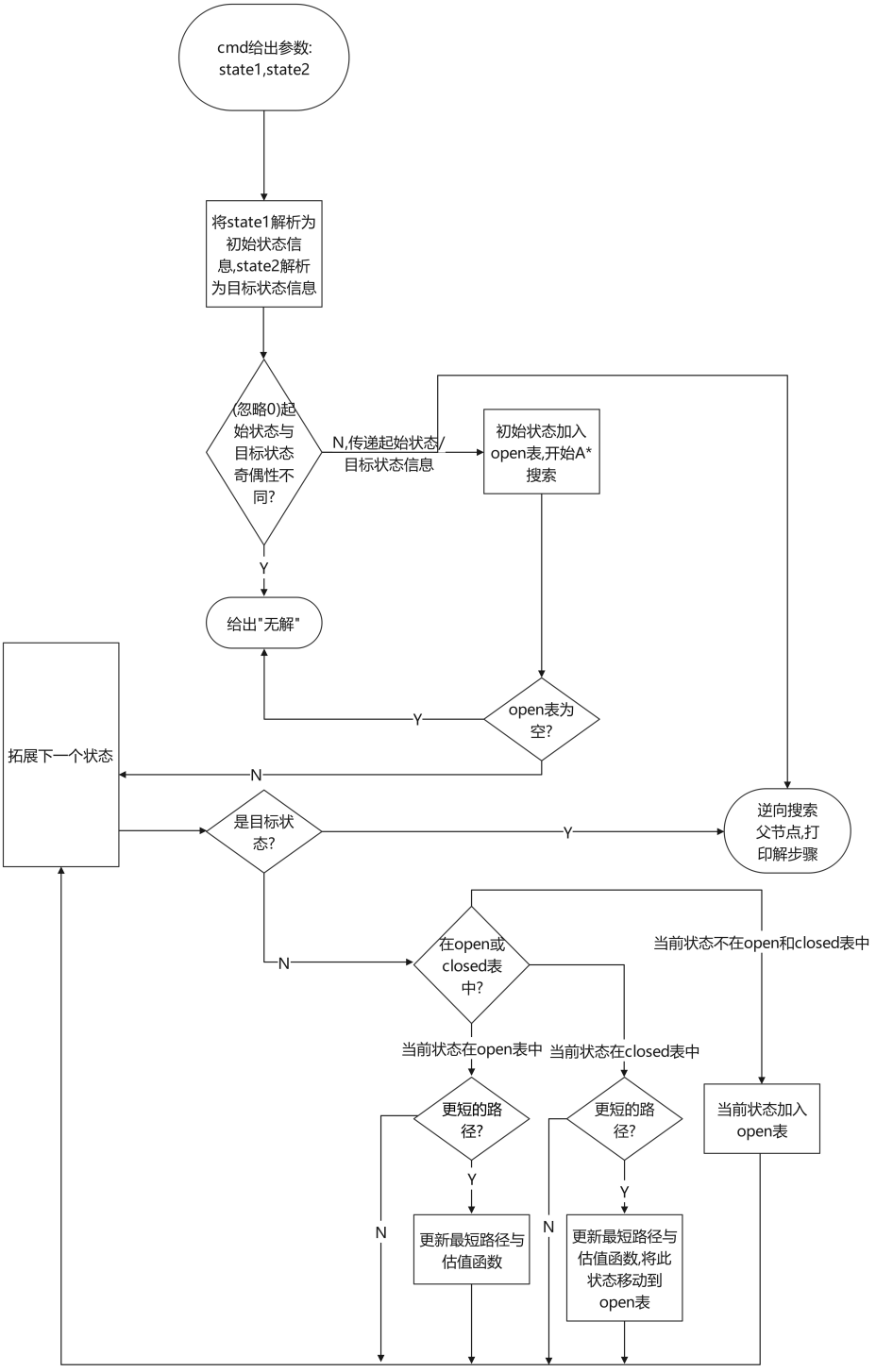

图 1 实验设计思路流程图

2.2 核心算法及基本原理

本次实验所用到的核心算法为A*算法。

A*算法是一种启发式搜索，是在静态路网中求解最短路径一种较为有效的直接搜索算法，实质上可以看成是在Dijkstra算法的基础上进一步进行剪枝后得到的一种较为高效的搜索方式。而在A*算法中，最为核心以及关键的一个公式可表示为：

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

其中，f(n)为从起始状态经过状态n到目标状态的总代价估计，g(n)为从起始状态到状态n之间的实际代价，而h(n)则为状态n到目标状态最佳路径的代价估计。而算法设计的关键，主要取决于估价函数h(n)的选取。当h(n)比实际距离小时，效率会偏低，但能保证找到最短路径，即最优解；而当h(n)比实际距离大时，效率高，但无法保证能否找到最优解。可见，h(n)的恰当选取是决定算法性能以及问题解决效率的主要因素。

2.3 模块设计

- 启发函数h1：用于统计不位于目标位置的棋子数目

```
def h1(s):
    a=0
    for i in range(len(s.node)):
        for j in range(len(s.node[i])):
            if s.node[i][j]!=goal.node[i][j]:
                a+=1
    return a
```

- 启发函数h2：用于计算所有棋子到其目标位置的距离之和

```
def h2(s):
    a=0
    for i in range(len(s.node)):
        for j in range(len(s.node[i])):
            tmp=s.node[i][j]
            for m in range(len(goal.node)):
                for n in range(len(goal.node[m])):
                    if tmp==goal.node[m][n]:
                        a=a+abs(m-i)+abs(n-j)
    return a
```

• 逆序数计算函数inverse_num：对起始状态和目标状态进行逆序数奇偶性计算

```
def inverse_num(M):
    a=0
    for i in range(len(M)):
        for j in range(len(M)):
            if M[i]!='0' and M[j]!='0' and i<j and M[i]>M[j]:
                a+=1
    a=a%2
    return a
```

• 逆序数判断函数Solve：对计算出的两个逆序数奇偶性结果进行比较

```
def Solve(A,B):
    a=1
    if(inverse_num(A)!=inverse_num(B)):
        a=0
        print('无解')
    return a
```

• 排序函数list_sort：对openlist中结点按照估价函数f的值进行排序

```
def list_sort(l):
    cmp=operator.attrgetter('f')
    l.sort(key=cmp)
```

• A*算法函数A_star1：整个文件中的核心函数，用于寻找最短路径

```
def A_star1(s):
    global openlist
    global closelist
    global A,B
    if Solve(A,B)==0:
        return None
    openlist=[s]
    while(openlist):
        get=openlist[0]
        if(get.node==goal.node).all():
            openlist.remove(get)
            closelist.append(get)
            return get
        openlist.remove(get)
        closelist.append(get)
        #判断此时空格位置(a,b)
        for a in range(len(get.node)):
            for b in range(len(get.node[a])):
```

```
                if get.node[a][b]==0:
                    break
            if get.node[a][b]==0:
                break

        #开始移动
        for i in range(len(get.node)):
            for j in range(len(get.node[i])):
                c=get.node.copy()
                if (i-a)**2+(j-b)**2==1:
                    #直接图进行操作
                    c[a][b]=c[i][j]
                    c[i][j]=0
                    #每个new为一个节点
                    new=State(c)
                    new.father=get
                    new.g=get.g+1#新节点与父亲节点的深度差1
                    new.h=h2(new)
                    new.f=new.g+new.h
                    print(new.node)
                    print(new.father.node)
                    print('----')
                    Add=1
                    for item in closelist:
                        if (item.node==new.node).all():
                            Add=0
                    if Add==1:
                        openlist.append(new)
        list_sort(openlist)
```

- 输出函数printpath：对于寻找到的最短路径进行输出

```
def printpath(f):
    if f is None:
        return
    printpath(f.father)
    print(f.node)
```

2.4 其他创新内容或优化算法

- 使用了Python自带的matplotlib,panda等库来绘制搜索生成树，部分代码如下：

```
get_ipython().run_line_magic('matplotlib', 'inline')
df = pd.DataFrame({ 'father':[str(i.node) for i in
list1],"son":[str(j.node) for j in list2]})
G=nx.from_pandas_edgelist(df, 'father', 'son')
plt.figure(figsize=(1.5*len(list1),1.5*len(list2)))
# with_labels是否显示标签，node_size节点大小，node_color节点颜色，node_shape
节点形状，alpha透明度，linewidths线条宽度
nx.draw(G, with_labels=True,
font_size=1.5*len(list1),node_size=1000*len(list1),
node_color="skyblue", node_shape="s", alpha=0.8, linewidths=5)
plt.savefig('searchTree.jpg')
plt.show()
```

3 实验过程

3.1 环境说明

- 操作系统：Windows 10
- 开发语言：Python
- 开发环境：Spyder (Python 3.7)
- 核心使用库：numpy,operator,matplotlib,pandas,networkx等

3.2 源代码文件清单

- A-8数码.py：唯一源文件，用于同时实现A*算法解决八数码问题，输出最短路径以及绘制搜索树几个功能。

3.3 实验结果展示

- 使用的测试案例：

初始状态：283164705　　目标状态：123804765

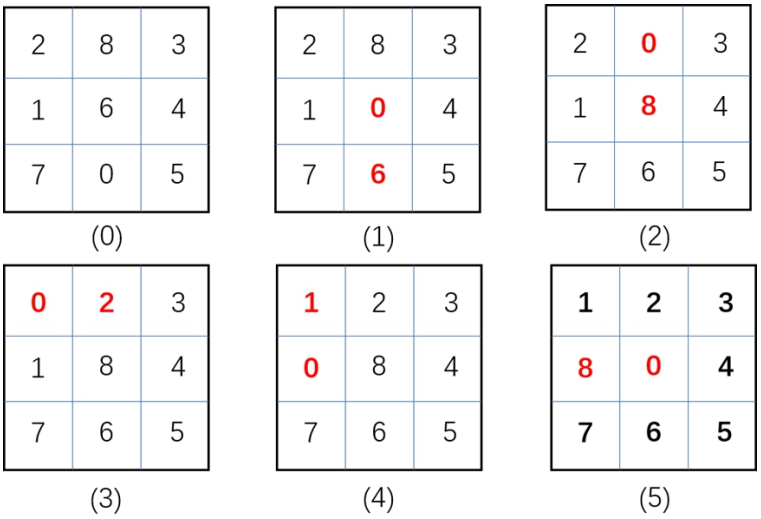

图 2 采用估价函数f1(n)得到的最短路径图

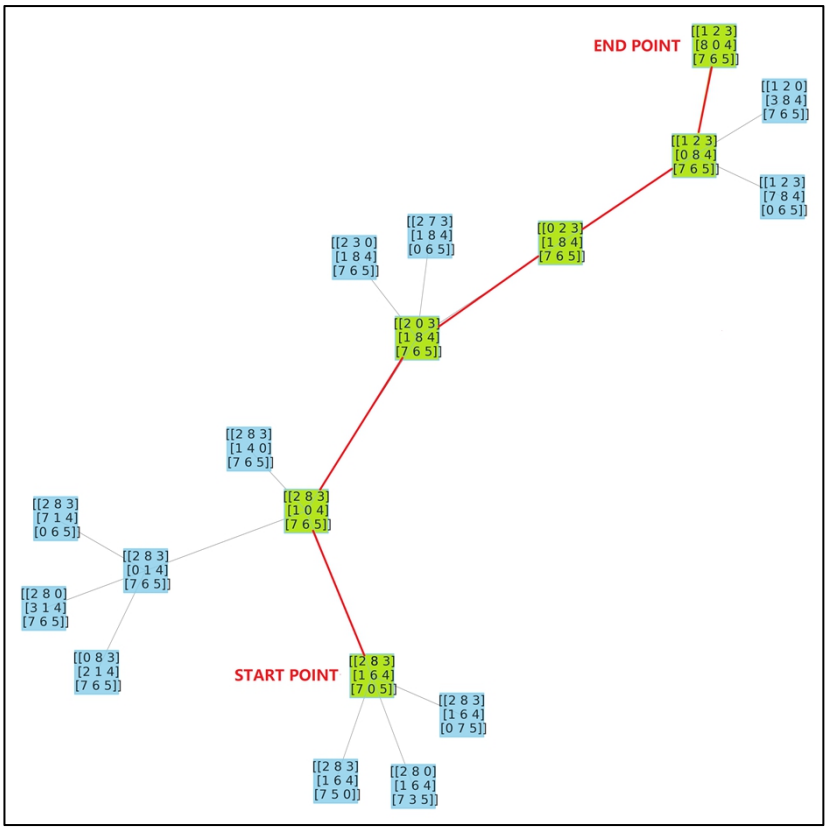

图 3 搜索树及最短路径

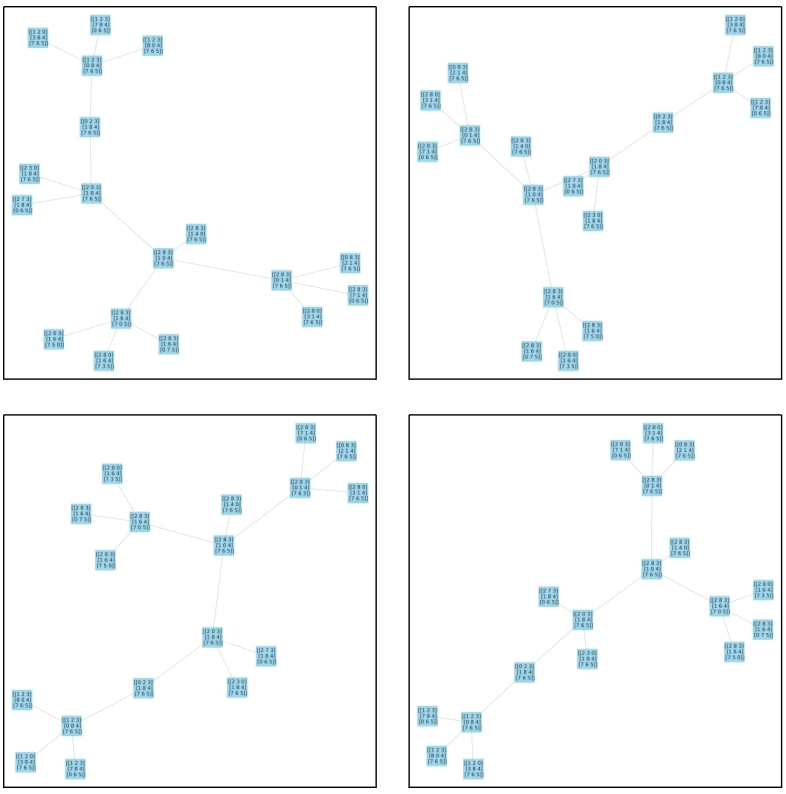

图 4 几种形式的搜索树

3.4 实验结论

针对A*算法使用不同的估价函数h(n)以及其他算法求解八数码问题的结果，可进行进一步的对比分析。此次对比，除了A*算法的两种估价函数以外，我们还引入了有界深度优先搜索以及广度优先搜索两种算法：

- **有界深度优先搜索(DFS)**

为了解决深度优先搜索不完备的问题，避免搜索过程陷入无穷分支的死循环，有界深度优先搜索方法被提出。有界深度优先搜索的基本思想是：对深度优先搜索方法引入搜索深度的界限，当搜索深度达到了深度界限，而尚未出现目标结点时，就换一个分支进行搜索。

- **广度优先搜索(BFS)**

其基本思想为：从起始状态结点开始，逐层地对结点进行扩展并考察它是否为目标结点，在第n层的结点没有全部扩展并考察完之前，不会对第n+1层的结点进行扩展。closelist中的结点总是按照先后顺序进行排列，即新的结点总是排在列表的后面。因此，只有某一层的结点全部被遍历过以后，才会开始进行对下一层结点的考察。

为了更有效地进行性能分析，我们重点比较了不同算法的几个指标，其中包括算法时长(time cost)、最大深度(maxDepth)、搜索节点数目以及有效分支因子b*。

所谓有效分支因子b*，是指对于某个问题，令A*算法生成的总结点数为N，解的深度为d。那么b*就是深度为d的标准搜索树为了能够包括这N+1个节点，所必需的分支因子。即

$$N + 1 = 1 + b^* + pow(b^*, 2) + pow(b^*, 3) + \ldots + pow(b^*, d)$$

不同算法对比结果如下：

表 1 算法性能对比

算法	time cost	maxDepth	搜索节点数目	有效分支因子b*
A*-h1=不在位的棋子数	0.996 ms	5	14	1.5
A*-h2=所有棋子到其目标位置的距离和	0.998 ms	5	12	1.5
有界DFS(maxDepth=8)	165.623 ms	8	246	2.0
BFS	3.985 ms	5	61	2.1

由实验数据可以看出，对于A*算法，使用估价函数h1，即统计不在目标位置的棋子数，消耗的时间比估价函数h2稍微少一些，即在效率上更具有优势。而A*算法与其他两种算法相比，效率又明显更高一些。

这是因为，有界深度优先搜索和广度优先搜索均为盲目搜索。它们的一个共同特点就是它们的搜索路线都是事先决定好的，是按照固定的顺序去进行搜索的，因而这样的搜索策略都具有较大的盲目性。盲目搜索所需扩展的结点数目很大，产生的无用结点肯定就很多，效率就会较低。

盲目搜索没有利用被求解问题的任何特征信息，在决定要被扩展的结点时，没有考虑该结点到底是否可能出现在解的路径上，也没有考虑它是否有利于问题的求解以及所求得的解是否为最优解（广度优先搜索能得到路径最优解，但不一定能得到代价最优解）。

而A*算法则是充分利用了待求解问题自身的某些特征信息，即每一个状态的估价函数，来指导搜索朝着最有利于解决问题的方向发展。在选择相应结点进行扩展时，就会优先选择那些代价最小的结点，此时搜索的效率就会大大提高。这种与A*算法具有类似特征，即利用问题自身信息来有效提高搜索效率的搜索策略，也被称作启发式搜索。

总之，在解决八数码问题的效率上，A*算法相较于其他算法来说，的确具有非常大的优势，这可以从该算法极高的效率体现出来。

4 总结

4.1 实验中存在的问题及解决方案

问题1：

如何能比较方便快捷地在找到最短路径的同时保存搜索的序列并绘制搜索树？

解决方案：

考虑利用Python语言里自带的库，比如matplotlib等，来直接绘制搜索生成的树。具体方法为在搜索过程中使用一个列表closelist，在寻找最短路径时按照先后顺序将遍历到的结点储存到closelist中。在成功找到最短路径之后，利用matplotlib和panda等库里的函数将closelist中的结点以树的形式绘制出来。

问题2：

如何设计不同的估价函数？

解决方案：

在经过查阅相关资料并进行讨论后，我们决定采用两种不同的估价函数，一种是统计不在目标位置的棋子数目；另外一种则比第一种要复杂一些，需要计算每一个不在目标位置的棋子相对其目标位置的曼哈顿距离，并求距离的总和。

问题3：

如何提高问题解决的效率？

解决方案：

在对八数码问题相关内容进行学习的过程中，我们了解到其状态序列逆序数的奇偶性对于问题可求解性的影响。具体来说，若起始状态与目标状态逆序数的奇偶性相同，则问题一定有解，反之问题一定无解。因此，可以利用这一推论，在进行A^*算法搜索之前，先对两个状态的逆序数奇偶性进行判断。若两者相异，则可直接得出无解的结论，无需再进行进一步的搜索，大大节省了解决问题的时间。

4.2 心得体会

在求解过程中，我们查阅了大量的资料，目的是对于A^*算法以及八数码问题的基本原理及内容有一个比较深入的了解，以便于我们进行算法设计。在设计过程中，我们也遇到了不少的问题，比如如何准确且直观地表示每一个状态，如何清楚展示最后找到的最短路径，以及如何保存每一次搜索的序列，从而绘制出最后的搜索树等等，最后也是通过不断讨论与探索才逐一解决了这些问题。总的来说，此次实验的确是一个综合性比较强的实验。

4.3 后续改进方向

- 对于输出结果的形式以及界面等等可以进行进一步优化，使其更加直观
- 对于估价函数的构造可以有其他尝试
- 对于搜索生成树的绘制方法也可以尝试其他方式

4.4 总结

此次实验主要内容为利用A*算法求解八数码问题，目的是希望我们在求解过程中进一步体会启发式搜索、估价函数以及A*算法的定义和内涵，并在设计算法过程中熟悉A*算法的思想以及具体实现方式。

实验的本质内容是设计A*算法以解决实际问题，但实际上，实验过程中需要解决的问题远不止这一个，还包括状态的表示方式，输出形式，以及搜索生成树的绘制方法等等，这些都是在本次实验中需要考虑并解决的问题。因此，在实验中，解决这些问题的过程，是比较考验我们各方面能力的。在查找讨论再到探究尝试的整个过程中，我们解决问题的能力得到了一定的锻炼。同时，对于A*算法的具体实现以及Python中绘图相关库的使用方法也有了更加深入的了解。可以说，有了不少的收获。

参考文献

[1]卜奎昊, 宋杰, 李国斌. 基于A*算法的八数码问题的优化实现[J]. 计算机与现代化, 2008(1):29-31.

[2]张信一, 黎燕. 基于A^*算法的八数码问题的程序求解[J]. 现代计算机:上下旬, 2003.

[3]欧阳林艳. 八数码问题的搜索算法比较. 洛阳师范学院学报, 2011.

成员分工

- 苗成林 1851804：负责代码实现以及搜索树的绘制
- 赵昊堃 1852877：负责流程图的绘制以及PPT制作
- 唐骏龙 1851785：负责论文撰写