西安电子科技大学

人工智能概论大作业

题　目：《基于图搜索算法的问题求解》

学 生 信 息 及 分 工 情 况：

编程：

21009200731 蔡明硕　　21009200780 付友泉

21200100001 王志强　　21009201019 李嘉辉

论文：

21009200696 杨子璇　21009200623 刘博玮　21009200732 李晶

21009201015 冯欣轶　21009200947 史子昂

基于状态空间搜索的 A*算法解决八数码问题

摘要

本文用人工智能领域中经典的**A*算法**解决了人工智能中常见的**八数码问题**。

本文首先介绍了**状态空间搜索**和八数码问题,并对 A*算法进行了解释。针对八数码问题，本文定义了灵活的**估价函数**,即取启发函数 h(n)=P(n)（P(n)定义为每一个数码与目标位置之间距离(不考虑夹在其间的数码)的总和），分析了估价函数对程序效率的影响,并推广了八数码问题的应用，具有一定的现实意义。

关键词：八数码 启发式搜索 状态空间搜索 A*算法

ABSTRACT

In this paper, the classical A * algorithm in the field of artificial intelligence is used to solve the common eight-digit problem in artificial intelligence. Firstly, this paper introduces the state space search and eight-digit problem, and explains the A * algorithm. For the eight-digit problem, a flexible evaluation function is defined, namely, the heuristic function H (N) = P (N)(P (N) is defined as the sum of the distances between each digit and the target position (regardless of the digits between them) , the influence of evaluation function on program efficiency is analyzed, and the application of eight-digit problem is extended, which has certain practical significance.

Key words: octagon，heuristic search，state space search，A* algorithm

目录

一．绪论

1.1 状态空间表示法

从求解现实问题的过程来看，人工智能的多个研究领域都可以抽象为一个“问题求解”的过程，而问题求解实际上就是一个搜索过程。在搜索过程开始之前，必须先用某种方法或某几种方法的混合来表示问题。问题的求解技术主要涉及两个方面：问题的表示和求解的方法。状态空间表示法就是用来表示问题及其搜索过程的一种方法。基于解答空间的问题表示和求解方法，是以状态和算符为基础来表示和求解问题的。主要包括三个要素：状态，算符，状态空间。

- 状态

状态是表示问题求解过程中每一步问题状况的数据结构，一般用一组数据表示：Sk=[Sk]$_0$，[Sk]$_1$……式中每个元素为集合的分量，称为状态变量：每一个分量给予确定的值时，会得到一个具体的状态；只要有利于问题求解，任何一种类型的数据结构都可以用来描述状态；在程序中，用字符、数字、记录、数组、结构、对象等表示状态。

- 算符

当对一个问题状态使用某个可用操作时，将引起该状态中某些分量值的变化，从而使问题从一个具体状态变为另一个具体状态。算符可以理解为状态集合上的一个函数，它描述了状态之间的关系；算符可以是某种操作、规则、行为、变换、算子、函数或过程等；算符也称为操作，问题的状态也只能经定义在其上的这种操作而改变。

- 状态空间

状态空间用来描述一个问题的全部状态以及这些状态之间的相互关系，状态空间常用一个三元组表示：（S，F，G）。其中，S 为问题的所有初始状态的集合，F 为算符的集合，G 为目标状态的集合。

1.2 八数码问题

八数码问题，也叫九宫问题，是人工智能中状态搜索中的经典问题。该问题的描述为：在 3×3 的棋盘中，摆有八个棋子，每个棋子上标有 1 至 8 的某一数字，不同棋子上标的数字不相同。棋盘上还有一个空格，与空格相邻的棋子可以移到空格中。要求解决的问题是：给出一个初始状态和一个目标状态，找出一种从初始转变成目标状态的移动棋子步数最少的移动步骤。

二．A*算法介绍

为了便于快速寻找最优解，A*算法需要对估价函数做出一定的限制。估价函数是一种用于估计节点重要性的函数。它通常被定义为从初始节点 S,出发，约束经过节点 n 到达目标节点 Sg 的所有路径中最小路径的估价值估价函数 f(n)的一般形式为

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

其中，g(n)为从初始节点 S。到约束节点 n 的实际代价，h(n)是从约束节点 n 到目标节点 Sg 的最优路径的估计代价。对于 g(n)的值，可以按指向父节点的指针，从约束节点 n 反向跟踪到初始节点 S。，

得到一条从初始节点 S。到约束节点 S。的最小代价路径,然后把这一条路径上的所有有向边的代价相加,就得到 g(n)的值。对 h(n)的值,需要根据问题自身的特性来确定,它体现的是问题自身的启发性信息,因此也称 h(n)为启发函数。记 g*(n)是从初始节点 S0 到任意节点的一条最佳路径的代价，h*(n)是从初始节点到该节点的一条最佳路径的代价。而估价函数 f(n)是 f*的一个估计，即

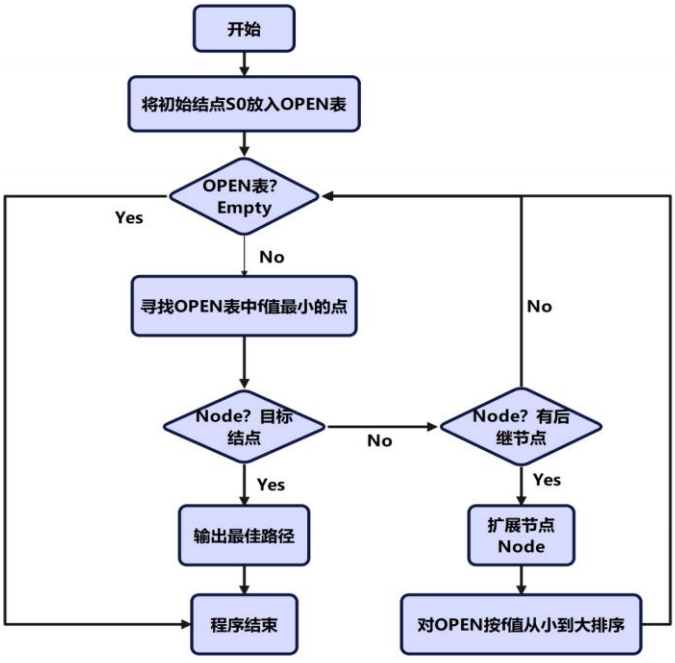
$$f(n)=g(n)+h(n)$$

其中 g(n)是 g*(n)的估计,h(n)是 h*(n)的估计。那么，对估价函数的限制就是以下两个条件:

(1)g(n)≥g*(n)

(2)h(n)是 h*(n)的下界，即对所有的节点 n 均有 h(n)≤h*(n)。满足以上约束的估价函数定义的 A 算法就是 A*算法。

A*算法的流程图如下：



三．实验过程

对于八数码问题，取启发函数 h(n)=P(n),P(n)定义为每一个数码与目标位置之间距离(不考虑夹在其间的数码)的总和，同样要判定至少要移动 P(n)步才能达到目标，因此有 P(n)≤h*(n),即满足 A*算法的限制条件。

针对八数码问题的 A*算法代码

```
#include<iostream>
#include<queue>
#include<map>
#include<stack>
using namespace std;

map<int, int>vis;
map<int, int>step;
map<int, int>id;
map<int, int>parent;
queue<int>open;
queue<int>close;
queue<int>nclose;
stack<int>out;
int m, n;
int dir[4][2] = { 0,-1,-1,0,0,1,1,0 };

int change(int**p)
{
int s = 0;
for(int i=0;i<m;i++)
for (int j = 0; j < n; j++)
s = s * 10 + p[i][j];
return s;
}

void getid(int a,int b)     //!! 获取与目标状态不同的格子数//h 函数,
{
int c;
c = a;
```

```
for (int i = 0; i < m * m; i++)
{
if ((a % 10) != (b % 10))
id[c]++;
a = a / 10, b = b / 10;
}
}

/*
循环一下步骤
1.取出 CLOSED 表中第一个元素（u），作为可扩展状态
2.计算该状态的每个可扩展状态的 h（n），将这些可扩展状态放入 open 表中
//h(n)的计算方式是与目标不同的格子数
3.找到 OPEN 表中全局 f（n）(f(n)=g(n)+h(n))最小的元素，放入 CLOSED 表中

有以下注意的:
当 CLOSED 中的第一个元素==目标状态时，停止搜索，返回(f(n)==g(n))
当 OPEN 表中为空时，停止循环，返回-1，表示无结果
会对每一个放入 CLOSED 中的元素进行标记，在以下搜索中若遇到已被标记的状态不进行搜索
*/
int A(int u, int v)     //启发式搜索
{
open.push(u);
vis[u] = 1;
getid(u, v);
while (open.size())
{
int q, p, w, x, y, newx, newy, size;
if (open.front() == v)    //找到目标状态
return step[v];
size = open.size();
for(int i=0;i<size;i++)    //找出该层数中，最小的 id 值
{
int** r;
w=q=open.front();   //取 CLOSED 表中第一个元素，该元素是上一步搜索的 f（n）最小的元素。进行扩展
open.pop();
r = new int* [n];
for (int i = 0; i < n; i++)
r[i] = new int[m];
for (int i = m - 1; i >= 0; i--)
for (int j = n - 1; j >= 0; j--)
{
```

```
r[i][j] = q % 10, q = q / 10;
if (r[i][j] == 0)//0 代表空格，在该步骤显示
{
x = i, y = j;
}
}
for (int i = 0; i < 4; i++)    //~~扩展
{
newx = x + dir[i][0], newy = y + dir[i][1];
if (newx >= 0 && newx < m && newy >= 0 && newy < n)   //若该位置可交换
{
r[x][y] = r[newx][newy];    //交换空白格位置
r[newx][newy] = 0;
p = change(r);
if (!vis[p])    //该状态没有被访问过%%%%%%访问过的不放入
{
close.push(p);     //把该状态放进 open 表
nclose.push(p);
vis[p] = 1;
step[p] = step[w] + 1;    //层数(g(n))在原来状态上加一
parent[p] = w;    //标记父状态
getid(p, v);
}
r[newx][newy] = r[x][y];    //变回原来状态
r[x][y] = 0;
}
}
}
if (close.size())   //若 open 表不为空
{
int csize = close.size(), min;
min = id[nclose.front()];
for (int i = 0; i < csize; i++)     //找出 open 表中 f（n)的最小值
if (id[nclose.front()] < min)
{
min = id[nclose.front()];
nclose.pop();
}
else nclose.pop();
for (int i = 0; i < csize; i++)     //把 open 表中 f(n)最小值的状态放进 CLOSED 表中
if (id[close.front()] == min)
{
open.push(close.front());
close.pop();
```

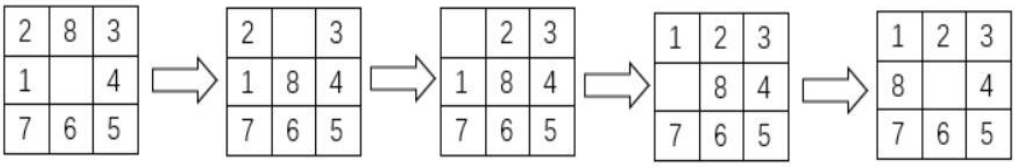
```
}
else close.pop();
}
}
return -1;
}

int main()
{
cout << "A 搜索" << endl;
int u, v, t;
int** mau, ** mav;
cout << "输入 m*n：" << endl;
cin >> m >> n;
mau = new int* [n], mav = new int* [n];
for (int i = 0; i < n; i++)
mau[i] = new int[m], mav[i] = new int[m];
cout << "输入初始状态：" << endl;
for (int i = 0; i < m; i++)
for (int j = 0; j < n; j++)
cin >> mau[i][j];
cout << "输入最终状态：" << endl;
for (int i = 0; i < m; i++)
for (int j = 0; j < n; j++)
cin >> mav[i][j];
u = change(mau), v = change(mav);
if (A(u, v) != -1)
{
cout << "到达目标状态需要 " << A(u, v) << " 步" << endl;//!!A 是需要多少步
A*=g(n)+h(n)
t = v;
while (t)
{
out.push(t);
t = parent[t];
}
while (out.size())   //输出到达目标状态的过程
{
int** o;
t = out.top();
out.pop();
o = new int* [n];
for (int i = 0; i < n; i++)
o[i] = new int[m];
```

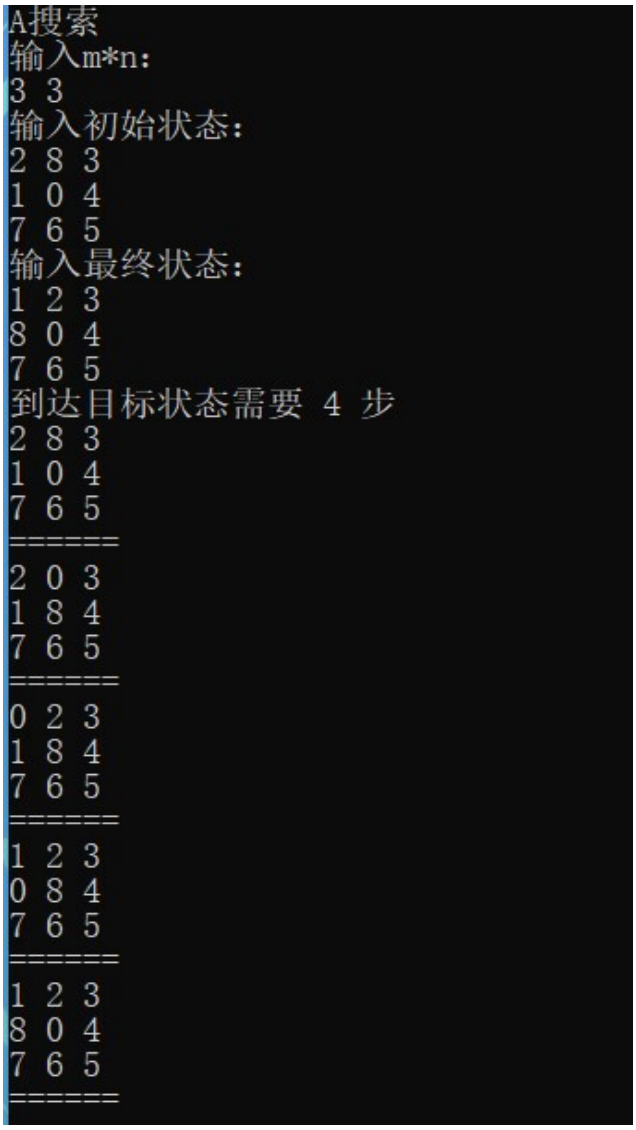
```
for (int i = m - 1; i >= 0; i--)
for (int j = n - 1; j >= 0; j--)
o[i][j] = t % 10, t /= 10;
for (int i = 0; i < m; i++)
{
for (int j = 0; j < n; j++)
cout << o[i][j] << " ";
cout << endl;
}
cout << "======" << endl;
}

}
else cout << "无解" << endl;
}
```

本文所用的八数码问题测试样例和算法流程如下：



程序运行结果如下：



通过对该测试样例的分析和运行，验证了本文程序的正确性，成功解决了八数码难题。

四．结语

八数码难题是人工智能的一个经典问题。在实验中，我们的程序在解决八数码难题方面，有着良好的执行效率和较低的时间复杂度。同时，我们的代码还对八数码问题进行了扩展，改进了 Open 表和 Closed 表，可以解决行数（m）>3 和列数（n）>3 时的数码难题。此外，八数码问题可以转化为实际的生活中的问题，例如组合优化问题

等，可以在一般情况下得出最优的路径，具有一定的应用价值和现实意义。

五．参考文献

[1]刘若辰，慕彩虹，焦李成，刘芳，陈璞花等编著，人工智能导论，北京，清华大学出版社，2021 年 9 月第一版

[2]胡敏杰，A*算法的探讨及其对八数码问题的实现，2005 年第 3 期 （总第 49 期）漳州师范学院学报(自然科学版)