

# 人工智能原理



## 第三章 搜索技术

### 3.1 搜索概念回顾

#### 3.2 启发式搜索

#### 3.3 广度优先

#### 3.4 深度优先

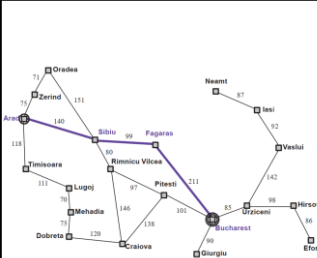
#### 3.5 启发式搜索

#### 3.6 遗传算法

#### 习题

### 3.1 搜索概念回顾

第三章 搜索技术



- 1、初始状态 (Initial state)
- 2、动作 (Actions)
- 3、转换模型 (Transition model)
- 4、目标测试 (Goal test)
- 5、路径代价 (Path cost)

### 3.1 搜索概念回顾

第三章 智能搜索

#### 1. 搜索的定义

求解问题的第一步，就是把问题描述清楚，也就是目标的表示，它涉及到一种知识表示策略——状态空间表示法。第二步就是搜索策略。这里的“搜索”是指，智能系统尝试性地找到目标解的动作序列。

搜索问题可分解为两个关键的子问题：（1）搜索什么；（2）在哪里搜索。前者是指搜索的解，即目标为何。后者是指搜索空间。搜索空间就是由一系列状态构成。那什么是状态呢？下面我们来讨论这个基础问题。

### 3.1 搜索概念回顾

第三章 智能搜索

#### 2. 状态空间表示

**定义1:** 状态 (State)：为描述不同事物之间的区别，而引入的最少一组变量  $q_0, q_1, \dots, q_n$  的有序结合，其形式如下：

$$Q = [q_0, q_1, \dots, q_n]$$

这里，变量  $q_i$  称为状态变量。给定某个分类的一组值，一个具体的状态就此确定下来。比如说，对于下棋而言，每一个静态的棋局都是一个状态，每走动一步，棋局就不一样，也就是状态就不一样。那么  $q_i$  就是棋局中的每个棋子所处的位置。

### 3.1 搜索概念回顾

第三章 智能搜索

#### 2. 状态空间表示

**定义2:** 状态空间 (State Space)：某个问题的全部可能状态或关系集合。

还是拿下棋来举例，它的状态空间，就是指它的每一个合法棋局的全体集合。很显然，由于状态空间可能非常巨大，所以在搜索之前，通常并不会一次性地把所有状态都生成出来，而是渐进式地扩展，“目标”状态就是在每次新展开的状态中搜索。

3.1 搜索概念回顾

第三章 智能搜索

2. 状态空间表示

定义2：状态空间（State Space）：某个问题的全部可能状态或关系集合。

和普通搜索算法不同的是，对于人工智能系统而言，在问题求解之前，搜索空间是未知的。通常是“走一步、看一步”，“摸着石头过河”。因此，搜索通常分为两个阶段：1) 状态空间的生成阶段。2) 对该状态空间中寻找目标解的阶段。对于博弈类游戏，上述特征尤其明显。我们通常无需把每一个棋局都考虑一边，而是在对方落子后，方才考虑我方可能走的每一步有利的棋局。

3.1 搜索概念回顾

第三章 智能搜索

2. 状态空间表示

定义3：操作算子（Operator）：使问题从一种状态变迁到另外一种状态的操作规则或函数。

$$F = \{f_0, f_1, \dots, f_m\}$$

操作算子可以是某个动作（如下棋的走步），也可以数学运算符、逻辑运算符等。

3.1 搜索概念回顾

第三章 智能搜索

2. 状态空间表示

有了上面概念上的铺垫，下面我们可以给出基于状态空间法的搜索算法基本流程：

1

• 定义状态空间。根据问题的特性，给出相应的状态空间，包括初始状态、目标状态和状态的一般表示形式。

2

• 确定操作算子集合：能够作用于一个状态后，迁移到另外一个状态。

3

• 确定一组搜索策略，使得能够从初始状态出发，沿着某条路径出发，达到目标路径。

3.1 搜索概念回顾

第三章 智能搜索

2. 状态空间表示

这样一来，问题求解的求解过程可归纳为：应用合法的规则和控制策略，取遍历或搜索状态空间，直至找到从初始状态到目标状态的某个路径。在这个过程中，要涉及两类函数：

(1) 目标检测函数：用来确定某个状态是不是目标状态。

(2) 路径代价函数：对每条路径赋予一定的权重，看走那条路最划算（代价最小）。

3.1 搜索概念回顾

第三章 搜索技术

3 举个例子：8滑块问题

• 状态：8个数字滑块每个占据一个方格，空格在最后

• 1、初始状态：不限

• 2、动作：上、下、左、右

• 3、转换模型：给定状态和动作，返回结果状态

• 4、目标测试：检查状态是否达到目标布局

• 5、路径代价：路径步数

1	2	3
8	6	4
	7	5

→

1	2	3
8		4
7	6	5

八滑块游戏初始与结果

3.1 搜索概念回顾

第三章 智能搜索

3. 状态空间表示

我们将九宫格中的格子从左到右，从上至下依次编号成1至9个格子，如图3-2所示。

1号	2号	3号
4号	5号	6号
7号	8号	9号

八数码游戏格子编号

2

## 3.1 搜索概念回顾

第三章 智能搜索

## 3. 状态空间表示

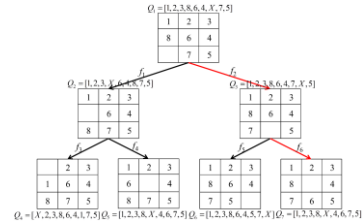
则我们的状态空间中的初始状态为:  $Q_1 = [1, 2, 3, X, 6, 4, 8, 7, 5]$ 。其中  $X$  代表该位置的数字为空。现在如图3所示, 有以下几种操作算子:

- $f_1$  = 数字8移动到X位上。产生对应的状态为:  $Q_2 = [1, 2, 3, X, 6, 4, 8, 7, 5]$ 。
- $f_2$  = 数字7移动到X位上。产生对应的状态为:  $Q_3 = [1, 2, 3, 8, 6, 4, 7, X, 5]$ 。
- $f_3$  = 数字1移动到X位上。产生对应的状态为:  $Q_4 = [X, 2, 3, 8, 6, 4, 1, 7, 5]$ 。
- $f_4$  = 数字6移动到X位上。产生对应的状态为:  $Q_5 = [1, 2, 3, 8, X, 4, 6, 7, 5]$ 。
- $f_5$  = 数字5移动到X位上。产生对应的状态为:  $Q_6 = [1, 2, 3, 8, 6, 4, 5, 7, X]$ 。
- $f_6$  = 数字6移动到X位上。产生对应的状态为:  $Q_7 = [1, 2, 3, 8, X, 4, 6, 7, 5]$ 。

## 3.1 搜索概念回顾

第三章 智能搜索

## 3. 状态空间表示



状态空间表示示例八: 数码游戏

则操作算子的集合  $F = \{f_1, f_2, f_3, f_4, f_5, f_6\}$ , 目标状态  $Q_7 = [1, 2, 3, 8, X, 4, 6, 7, 5]$ 。

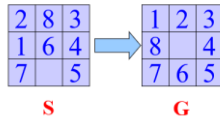
## 3.1 搜索概念回顾

第三章 智能搜索

## 3. 状态空间表示

## • 状态表示

- 用二维数组  $S[i, j]$  表示,
- 其中  $0 \leq i, j \leq 2$ ; 空格用0表示。
- 则:  $S[i, j] \in \{0, 1, 2, 3, 4, 5, 6, 7, 8\}$
- 用  $i_0$  和  $j_0$  表示空格的下标。



## • 定义操作符 (产生式规则):

- 左移  $R_1$  if ( $j_0 \geq 1$ ) then  $\{S[i_0, j_0] = S[i_0, j_0 - 1]; S[i_0, j_0 - 1] = 0;\}$
- 右移  $R_2$  if ( $j_0 \leq 1$ ) then  $\{S[i_0, j_0] = S[i_0, j_0 + 1]; S[i_0, j_0 + 1] = 0;\}$
- 上移  $R_3$  if ( $i_0 \geq 1$ ) then  $\{S[i_0, j_0] = S[i_0 - 1, j_0]; S[i_0 - 1, j_0] = 0;\}$
- 下移  $R_4$  if ( $i_0 \leq 1$ ) then  $\{S[i_0, j_0] = S[i_0 + 1, j_0]; S[i_0 + 1, j_0] = 0;\}$

## 第三章 搜索技术

## 3.1 搜索概念回顾

## 3.2 启发式搜索

## 3.3 广度优先

## 3.4 深度优先

## 3.5 启发式搜索

## 3.6 遗传算法

## 习题

## 3.3 启发式搜索

第三章 智能搜索

## 1. 启发式搜索策略



- 或者说: 如何驱动机器人达成我的目标?
- 等同于问: 如何管理团队/企业/军队?
- 古语说: 赏罚分明
- 赏: 告诉机器人怎么做是对的——启发函数  $h(x)$
- 罚: 告诉机器人怎么做是不对的——损失/评价函数  $g(x)$
- 赏罚分明  $f(x)$ :

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

启发式搜索(Heuristically Search)又称为有信息搜索(Informed Search), 它是利用问题拥有的启发信息来引导搜索, 达到减少搜索范围、降低问题复杂度的目的, 这种利用启发信息的搜索过程称为启发式搜索。

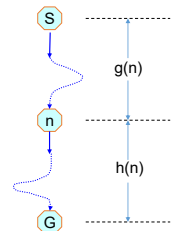
## 3.3 启发式搜索

第三章 智能搜索

## 1. 启发式搜索策略

## • 估价函数 (evaluation function) 与启发函数 (heuristic function)

- 利用启发信息, 构造一个函数, 对搜索路径全程的代价或希望进行评估。
- 如右图, 当前节点为  $n$ , 定义估价函数  $f(n)$ : 从初始节点  $S$  开始, 约束通过  $n$ , 到达目标节点  $G$  的全程代价的估计值。即:  $f(n) = g(n) + h(n)$ 
  - ✓ 其中:  $g(n)$  为从  $S$  到  $n$ , 已经走过的路径代价估计, 通常即用实际花费代价, 也比较容易计算。
  - ✓  $h(n)$  是从  $n$  到达目标  $G$  的代价的估计, 这一段路径是没有走过的, 必须根据问题特性, 利用启发信息进行估算。搜索的启发性即体现在  $h(n)$  上, 所以把  $h(n)$  叫做启发函数



3.3 启发式搜索

第三章 智能搜索

1. 启发式搜索策略

1. 启发信息

关于问题领域的，用来帮助搜索的信息。

2. 2. 启发信息按用途分类

用于决定下一个要扩展的节点

总是选择最有希望产生目标的节点（邻接点）优先扩展，即OPEN表按此希望值排序。这类启发信息使用最为广泛。

用于决定产生哪些子节点

扩展一个节点时，有选择的生成子节点（选择访问邻接点），有些明显无用或没有优势的子节点不令其产生出来。

用于决定从搜索图中修剪或抛弃那些节点

减小待搜索空间。

搜索图中，不访问这些顶点，或直接删除掉这些顶点。

3.3 启发式搜索

第三章 智能搜索

1. 启发式搜索策略

启发式搜索就是在状态空间中的搜索对每一个搜索的位置进行评估，得到最好的位置，再从这个位置进行搜索直到目标。这样可以省略大量无谓的搜索路径，提高了效率。

如果能够利用搜索过程所得到的问题自身的一些特征信息来指导搜索过程，则可以缩小搜索范围，提高搜索效率。像这样利用问题自身特征信息来引导搜索过程的方法成为启发式方法。

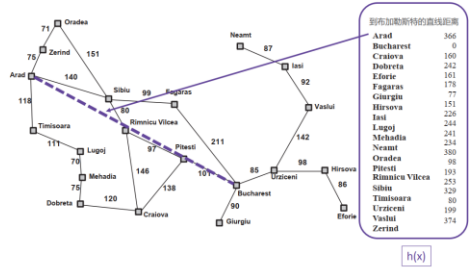
启发式策略可以通过指导搜索向最有希望的方向前进，降低了复杂性。通过删除某些状态及其延伸，启发式算法可以消除组合爆炸，并得到令人能接受的解(通常并不一定是最优解)。

3.3 启发式搜索

第三章 智能搜索

贪婪搜索举例

$f(n) = g(n) + h(n), g(n) = 0$

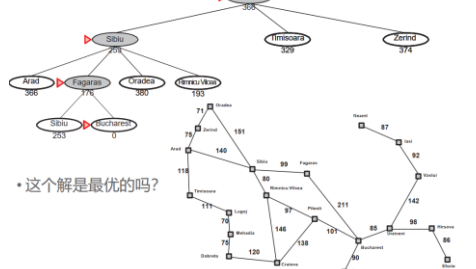


城市	到布加勒斯特的直线距离
Arad	366
Bucharest	0
Cluj	160
Dobru	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urechea	80
Vaslui	199
Zerind	374

3.3 启发式搜索

第三章 智能搜索

贪婪搜索举例



\* 这个解是最优的吗？

3.3 启发式搜索

第三章 智能搜索

1. 启发式搜索策略

然而，启发式策略是极易出错的。在解决问题的过程中启发仅仅是下一步将要采取的一个猜想，常常根据经验和直觉来判断。由于启发式搜索只有有限的信息(比如当前状态的描述)，要想预测进一步搜索过程中状态空间的具体行为则很难。一个启发式搜索可能得到一个次优解，也可能一无所获。这是启发式搜索固有的局限性。这种局限性不可能由所谓更好的启发式策略或更有效的搜索算法来消除。

一般说来，启发信息越强，扩展的无用节点就越少，引入强的启发信息，有可能大大降低搜索工作量，但不能保证找到最小耗散值的解路径(最佳路径)。因此，在实际应用中，最好能引入降低搜索工作量的启发信息而不牺牲找到最佳路径的保证。

3.3 启发式搜索

第三章 智能搜索

1. 启发式搜索策略

以下为一个启发式搜索的八数码游戏例子：

如图3-7所示，在一个九宫格里放入8个数字，数字只能上下左右移动，并且只能移动到空白处。通过若干次这样的移动后，把左面数字位置移动到右面数字位置。

2	8	3
1	6	4
7		5

→

1	2	3
8		4
7	6	5

八数码游戏初始与结果

3.3 启发式搜索

第三章 智能搜索

1. 启发式搜索策略

解决此问题的启发策略：每次移动的时候，正确位置数码的个数要大于交换前正确位置数码个数。正确位置数码的个数是指每个数码的位置与最终格局的对比，如果位置相同，则说明此数码在正确位置。图3-8中红色字体标识的数码为正确位置数码，由此我们可以发现下图中左边初始图案正确位置的数码个数为4个。

2	8	3
1	6	4
7		5

1	2	3
8		4
7	6	5

八数码游戏寻找正确位置数码个数

3.3 启发式搜索

第三章 智能搜索

1. 启发式搜索策略

由下图3-9所示可得，正确位置数码个数大于等于4的只有左下方的格局，那么下一步选择的的就是左下方的格局。

2	8	3
1	6	4
7		5

2	8	3
1	6	4
7	6	5

2	8	3
1	6	4
7	5	

2	8	3
1	6	4
	7	5

八数码游戏

3.3 启发式搜索

第三章 智能搜索

1. 启发式搜索策略

再次调用次算法如下图3-10所示：

2	8	3
1		4
7	6	5

2	8	3
	1	4
7	6	5

2	8	3
1	4	
7	6	5

2	8	3
1	8	4
7	6	5

八数码游戏

这样一步一步地进行，最终即可得到最终格局。

3.3 启发式搜索

第三章 智能搜索

2. 有序搜索 (A搜索)

$f(n) = g(n) + h(n), g(n) > 0, h(n) > 0$

有序搜索 (Ordered Search) 又称之为最佳优先搜索 (Best First Search)，是一种启发式搜索算法，我们也可以将它看作广度优先搜索算法的一种改进；最佳优先搜索算法在广度优先搜索的基础上，用启发估价函数对将要被遍历到的点进行估价，然后选择估价小的进行遍历，直到找到目标节点或者遍历完所有的点，算法结束。

在图搜索算法中，如果能在搜索的每一步都利用估价函数  $f(n) = g(n) + h(n)$  对Open表中的节点进行排序，则该搜索算法为A算法。

3.3 启发式搜索

第三章 智能搜索

2. 有序搜索

要实现最佳优先搜索我们必须使用一个优先队列 (Priority Queue) 来实现，通常采用一个 open 优先队列和一个 closed 集 open 优先队列用来储存还没有遍历将要遍历的节点，而 closed 集用来储存已经被遍历过的节点。我们用下面图3-11作为一个示例图来描述最佳优先搜索过程。

最佳优先搜索示例图

3.1 搜索概念回顾

第三章 智能搜索

2. 有序搜索

- 1. 扩展(Expanding)节点
  - 对某一节点 (状态)，选择合适的操作符作用在节点上，使产生后继状态 (子节点) 的操作。
  - 类似数据结构中的寻找邻接点，但这里的邻接点是选择操作后产生的。
- 2. Open和Closed表
  - 各种搜索算法在探索树的方式上不同，但都需要维护这两个表
  - Open表 (开放列表) 存放树中所有等待探索 (或扩展) 的点
  - Closed表 (封闭列表) 存放所有已经探索过的节点和不再考虑的节点。

3.1 搜索概念回顾

第三章 智能搜索

2. 有序搜索

- 两个表的结构可以相同，大致如下表：  
可根据需要扩展表的结构，比如加入代价字段等。  
在数据结构中，常用数组visited[]标记结点是否已经访问。

Open和Closed表结构示例

编号	状态节点	父节点

3.1 搜索概念回顾

第三章 智能搜索

2. 有序搜索

图搜索 (graph search) 一般过程：

- 建立一个只含初始状态节点S的搜索图G，建立一个OPEN表，用来存放未扩展节点，将S放入OPEN表中；
- 建立一个CLOSED表，用来存放已扩展节点和不再考虑的节点，初始为空；
- LOOP：若OPEN为空，则失败、退出；
  - 选择OPEN表中的第一个节点，将其移到CLOSED表中，称此节点为n节点；
  - 若n为目标节点，则成功、退出；（此解是追踪图G沿着指针从n到S这条路径得到的。）
  - 扩展n节点，生成不是n的祖先的那些后继节点的集合M。如果没有后继节点，则转LOOP
  - 把那些不在G中的M的成员作为n的后继节点加入G，并设置一个通向n的指针，把它们加入OPEN表。对已在G中的M的成员，调整有关指针
  - 按某一控制策略，重新排序OPEN表；

goto LOOP

3.3 启发式搜索

第三章 智能搜索

2. 有序搜索

【例】下图为一个城市地图，用最好优先搜索求从 a城市出发到达 i 城市的路径。  
g(n)用从a城市到n城市走过的实际距离。h(n)用后面的表给出，您可以设想h(n)为n到达i的直线距离。

评估函数值 $f(n)=g(n)+h(n)$

当前城市	h(n)
a	366
b	374
c	329
d	244
e	253
f	178
g	193
h	98
i	0

3.3 启发式搜索

第三章 智能搜索

2. 有序搜索

当前城市	h(n)
a	366
b	374
c	329
d	244
e	253
f	178
g	193
h	98
i	0

OPEN表	当前扩展	CLOSED表
a(366)	a	a
ab(449),ac(447),ae(393)	e	a, e
ab(449),ac(447),ae(440),aef(417),aeg(413)	g	a, e, g
ab(449),ac(447),ae(440),aef(417),aeg(415)	h	a, e, g, h
ab(449),ac(447),ae(440),aef(417),aeg(418)	f	a, e, g, h, f
ab(449),ac(447),ae(440),aef(417),aeg(418),aef(450)	i	a, e, g, h, f, i

3.3 启发式搜索

第三章 智能搜索

2. 有序搜索

分析：本题有3条解路径：  
aei, 全程实际代价为440；  
aefi, 全程实际代价为450；  
aeghi, 全程实际代价为418。  
A搜索算法得到的解为aeghi，本题中为最优解，因为本题满足了A算法的条件。  
但A算法不保证能取得全局最优解。

3.3 启发式搜索

第三章 智能搜索

2. 有序搜索

【例3.5.2】A算法求解8数码问题  
解：定义评估函数  $f(n)=g(n)+h(n)$   
 $g(n)$ =节点深度  
 $h(n)$ =当前盘面与目标对比,错位数字的个数  
如初始状态下:  $f(S)=g(S)+h(S)=0+3=3$

2	8	3
1		4
7	6	5

S

→

1	2	3
8		4
7	6	5

G

3.3 启发式搜索

第三章 智能搜索

2. 有序搜索

3.3 启发式搜索

第三章 智能搜索

3. 爬山搜索

$$f(n) = g(n) + h(n), g(n) = 0$$

1. 什么是爬山搜索?  
评估函数 $f(n)=g(n)+h(n)$ 中, 令 $g(n)=0$ , 即 $f(n)=h(n)$ , A 算法即成为爬山搜索算法。

2. 爬山搜索优点  
搜索效率高

3. 爬山搜索问题  
常常陷入局部最优, 而失去全局最优解。  
爬山搜索是一种贪婪(greedy)算法

3.3 启发式搜索

第三章 智能搜索

3. 爬山搜索

$$f(n) = g(n) + h(n), g(n) = 0$$

当前城市	h(n)
a	366
b	374
c	329
d	244
e	253
f	178
g	193
h	98
i	0

3.3 启发式搜索

第三章 智能搜索

3. 爬山搜索

$$f(n) = g(n) + h(n), g(n) = 0$$

解:  $f(n)=h(n)$ , 查表获得评估函数值。搜索过程见下表。解为: aei

当前城市	h(n)
a	366
b	374
c	329
d	244
e	253
f	178
g	193
h	98
i	0

OPEN表	当前扩展	CLOSED表
a(366)	a	a
ab(374), ac(329), ae(253)	e	a, e
ab(374), ac(329), aei(0), aef(178), aeg(193)	i	a, e, i

3.3 启发式搜索

第三章 智能搜索

4. 等代价搜索

$$f(n) = g(n) + h(n), h(n) = 0$$

1. 什么是等代价搜索?  
评估函数 $f(n)=g(n)+h(n)$ 中, 令 $h(n)=0$ ,  $f(n)=g(n)$ , 且 $g(n)$ 为实际路径代价, 此时的A算法称为等代价搜索。

2. 为什么属于盲目搜索?  
因为 $h(n)=0$ , 没有了启发性, 所以属于盲目搜索。

3. Prim算法和Dijkstra算法  
在图算法中经常要执行遍历每个顶点和每条边的操作, 即图搜索。许多图算法都以图搜索为基础, 如2-着色问题。连通性计算基于深度优先搜索 (DFS), 而无权最短路径则基于广度优先搜索 (BFS)。  
基于搜索的算法还包括计算最小生成树的Prim算法以及计算最短路径的Dijkstra算法。其中我们在前面章节已经介绍过广度优先搜索以及深度优先搜索, 本节着重介绍Prim算法和Dijkstra算法。

3.3 宽度优先 (BFS)

第三章 搜索技术

4. 等代价搜索

策略: 优先展开成本最低的节点

3.3 启发式搜索

第三章 智能搜索

4. 通用图搜索算法

普里姆 (Prim) 算法，图论中的一种算法，可在加权连通图里搜索最小生成树，即由此算法搜索到的边子集所构成的树中，不但包括了连通图中的所有顶点 (Vertex)，且使得树中所有的边的权值之和亦为最小值。  
Prim 算法基于贪心算法设计，该算法从一个顶点出发，选择这个顶点发出的边中权重最小的一条加入最小生成树中，随后从当前的树中的所有顶点发出的边中选出权重最小的一条加入树中，以此类推，直到所有顶点都在树中，算法结束。算法可以按如下具体描述：

3.3 启发式搜索

第三章 智能搜索

4. 通用图搜索算法

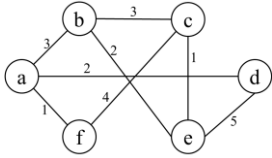
- 1). 输入：一个加权连通图，其中顶点集合为  $V$ ，边集合为  $E$ ；
- 2). 初始化：  $V_{new} = \{x\}$ ，其中  $x$  为集合  $V$  中的任一节点（起始点）， $E_{new} = \{\}$ ，为空；
- 3). 重复下列操作，直到  $V_{new} = V$ ；
  - a. 在集合  $E$  中选取权值最小的边  $\langle u, v \rangle$ ，其中  $u$  为集合  $V_{new}$  中的元素，而  $v$  不在  $V_{new}$  集合当中，并且  $v \in V$ （如果存在有多条满足前述条件即具有相同权值的边，则可任意选取其中之一）；
  - b. 将  $v$  加入集合  $V_{new}$  中，将  $\langle u, v \rangle$  边加入集合  $E_{new}$  中；
- 4). 输出：使用集合  $V_{new}$  和  $E_{new}$  来描述所得到的最小生成树。

3.3 启发式搜索

第三章 智能搜索

3. 通用图搜索算法

下面举一个例子来说明，图3-13是一个无向图，假设我们从顶点  $a$  出发使用 Prim 算法计算最小生成树，其算法运行过程如下。



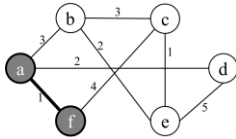
Prim算法示例图

3.3 启发式搜索

第三章 智能搜索

4. 通用图搜索算法

- ① 从顶点  $a$  发出的边有  $\langle a, b \rangle$ 、 $\langle a, d \rangle$  和  $\langle a, f \rangle$ ，其中权重最小的边为  $\langle a, f \rangle$ ，于是我们将边  $\langle a, f \rangle$  加入到最小生成树中，此时最小生成树包括图3-14中的阴影边和灰色顶点。



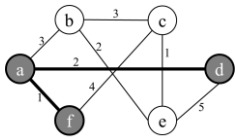
Prim算法示例图

3.3 启发式搜索

第三章 智能搜索

4. 通用图搜索算法

- ② 接下来我们继续从当前最小生成树中的顶点发出的所有边中寻找权重最小的一条，即边  $\langle a, b \rangle$ 、 $\langle a, d \rangle$ 、 $\langle f, c \rangle$  中的边  $\langle a, d \rangle$ ，于是我们将边  $\langle a, d \rangle$  加入到树中，如下图3-15所示。



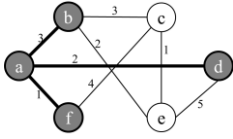
Prim算法示例图

3.3 启发式搜索

第三章 智能搜索

4. 通用图搜索算法

- ③ 继续上述步骤，从顶点  $a, f, d$  发出的边中选出权重最小的一条，即边  $\langle a, b \rangle$ ，并将它加入到树中，如下图3-16所示。



Prim算法示例图



3.3 启发式搜索

第三章 智能搜索

4. 通用图搜索算法

重复上述步骤，最后得到图的最小生成树如下图3-17所示。

Prim算法示例图

3.3 启发式搜索

第三章 智能搜索

4. 通用图搜索算法

迪杰斯特拉 (Dijkstra) 算法是由荷兰计算机科学家狄克斯特拉于1959年提出的，因此又叫狄克斯特拉算法。是从一个顶点到其余各顶点的最短路径算法，解决的是有向图中最短路径问题。迪杰斯特拉算法主要特点是以起始点为中心向外层层扩展，直到扩展到终点为止。

Dijkstra算法采用的是一种贪心的策略，声明一个数组  $Dis$  来保存源点到各个顶点的最短距离和一个保存已经找到了最短路径的顶点的集合  $T$ 。初始时，原点  $s$  的路径权重被赋为  $0(Dis[s]=0)$ 。若对于顶点  $s$  存在能直接到达的边  $(s, m)$ ，则把  $Dis[m]$  设置为  $w(s, m)$ ，同时把所有其他 ( $s$  不能直接到达的) 顶点的路径长度设为无穷大。初始时，集合  $T$  只有顶点  $s$ 。

3.3 启发式搜索

第三章 智能搜索

4. 通用图搜索算法

从  $Dis$  数组选择最小值，则该值就是源点  $s$  到该值对应的顶点的最短路径，并且把该点加入到  $T$  中，完成一个顶点。

然后，我们需要看看新加入的顶点是否可以到达其他顶点并且看看通过该顶点到达其他点的路径长度是否比源点直接到达短，如果是，那么就替换这些顶点在  $Dis$  中的值。

之后，从  $Dis$  中找出最小值，重复上述动作，直到  $T$  中包含了图的所有顶点。

3.3 启发式搜索

第三章 智能搜索

4. 通用图搜索算法

下面用一个示例图作解释，求图中从顶点  $a$  到其他各个顶点的最短路径。

Dijkstra算法示例图

3.3 启发式搜索

第三章 智能搜索

4. 通用图搜索算法

OPEN表	当前扩展	CLOSED表
a(0)	a	a
ab(75),ac(118),ae(140)	b	a,b
ac(118),ae(140)	c	a,b,c
ae(140),acd(229)	e	a,b,c,e
acd(229),aeg(220),aef(239),aei(440)	g	a,b,c,e,g
acd(229), aef(239),aei(440),aegh(317)	d	a,b,c,e,g,d
aef(239),aei(440),aegh(317)	f	a,b,c,e,g,d,f
aei(440),aegh(317),aefi(450)	h	a,b,c,e,g,d,f,h
aei(440),aefi(450),aeghi(418)	i	a,b,c,e,g,d,f,h,i

3.3 启发式搜索

第三章 智能搜索

4. 通用图搜索算法: Prim算法和Dijkstra算法区别

Prim更新的是**未标记集合到已标记集合之间的距离**

Dijkstra更新的是**源点到未标记集合之间的距离**

Prim是计算**最小生成树**的算法，比如为  $N$  个村庄修路，怎么修花销最少。

Dijkstra是计算**最短路径**的算法，比如从  $a$  村庄走到其他任意村庄的距离。

Prim

```
for k in range(n): #更新lowcost
    if (lowcost[k] > graph[v][k]):
        lowcost[k] = graph[v][k]
    w[k] = v #记录所有被更新的记录，新把当前点作为被更新权值的那条边的起始点
```

Dijkstra

```
for k in range(n):
    if (graph[v][k] < MAX and sign[k] and graph[v][k] + dict[v][k] < dict[k]):
        dict[k] = graph[v][k] + dict[v]
        parent[k] = v
```

3.3 启发式搜索

第三章 智能搜索

5. A\* 算法

约束最小代价 $f^*(n)$

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

为从初始状态S开始约束经过结点n, 到达目标结点G,  
**所有解路径上的实际最小路径代价。**  
 $g^*(n)$ 为从S到n的实际最小代价  
 $h^*(n)$ 为从n到G的估计的实际代价,  
若有多个目标结点,  $h^*(n)$ 取其中的最接近值。

3.3 启发式搜索

第三章 智能搜索

5. A\* 算法

A\*算法, A\* (A-Star)算法是一种静态路网中求解最短路径最有效的直接搜索方法, 也是解决许多搜索问题的有效算法。算法中的距离估算值与实际值越接近, 最终搜索速度越快。但是要注意的是, A\*算法是最有效的直接搜索算法, 之后涌现了很多预处理算法 (如ALT, CH, HL等等), 在线查询效率是A\*算法的数千至上万倍。A\*算法的公式表示为:

$$f(n) = g(n) + h(n), g(n) \geq g^*(n), h(n) \leq h^*(n)$$

其中,  $f(n)$ 是从初始状态经由状态n 到目标状态的代价估计,  $g(n)$ 是在状态空间中从初始状态到状态n 的实际代价,  $h(n)$ 是从状态n 到目标状态的最佳路径的估计代价。对于路径搜索问题, 状态表示为下文图中的节点, 代价则用距离来表示。而要算法可以保证找到最短路径 (最优解), 关键在于估价函数 $h(n)$ 的选取 (或者说 $h(n)$ 的选取)。

3.3 启发式搜索

第三章 智能搜索

5. A\* 算法

如下就A\*算法与深度优先搜索算法和广度优先搜索算法进行比较:  
深度优先搜索会朝一个方向进发, 直到遇到边界或者障碍物, 才回溯。一般在实现的时候, 我们采用递归的方式进行, 也可以采用模拟压栈的方式来实现。

3.3 启发式搜索

第三章 智能搜索

5. A\* 算法

如图3-24, S代表起点, E代表终点。如果我们按照右、下、左、上这样的扩展顺序的话, 算法就会一直往右扩张, 直到走到地图的右边界, 发现没找到目标点, 然后再回溯。

深度优先搜索方式

3.3 启发式搜索

第三章 智能搜索

5. A\* 算法

这个算法的好处就是实现简单, 不过也存在两个明显的问题: 1、路径可能不是最优解; 2、寻路时间比较长。  
图3-25展示了广度优先搜索的方式, 广度优先搜索像是地震波, 从起点向外辐射, 直到找到目标点。我们在实现的时候, 一般采用队列来实现。

广度优先搜索方式

3.3 启发式搜索

第三章 智能搜索

5. A\* 算法

广度优先搜索的优点是实现简单, 同时保证算法能够找到一条最优的路径。但是也存在不足之处就是算法消耗的时间比较大, 遍历的点会很多。那么这里我们就可以思考一个问题: 为什么广度优先算法能找到最优路径, 但是却很耗时呢?  
广度优先搜索之所以能找到最优的路径, 原因就是每一次扩展的点, 都是距离出发点最近、步骤最少的。如此这样递推, 当扩展到目标点的时候, 也是距离出发点最近的。这样的路径自然形成了最短的路线。  
正是由于广度优先搜索是一层层的扩展, 让它可以保证了算法能找到一条最优的路线的可能性, 但是却同时也因此消耗了更多的时间和计算能力去走了绝大多数的无效步骤。

3.3 启发式搜索

第三章 智能搜索

5. A\* 算法

从另一个角度看，广度优先搜索算法只关注了当前扩展点和出发点之间的关系，而忽略了当前点和我们的目标点之间的关系，也就是一种缺乏指引搜索，较为盲目的搜索方式。情况如图3-26所示时：

存在两种搜索方式时

3.3 启发式搜索

第三章 智能搜索

5. A\* 算法

同样是从出发点S走了两步以后，光标到达的  $M_1$  和  $M_2$  两个位置，如果让你来选择，你会选择他们中的谁来做扩展点呢？很明显，只要是眼力不差的人，都会选择  $M_1$ 。为什么呢？因为  $M_2$  需要再走9步，才能到达终点E；而  $M_1$  只需要7步。我们的判断依据，除了考虑了中间这个点同出发点的距离以外，还考虑了这个点同目标点的距离。A\*算法相对广度优先搜索算法，除了考虑中间某个点同出发点的距离以外，还考虑了这个点同目标点的距离。这就是A\*算法比广度优先算法智能的地方。

3.3 启发式搜索

第三章 智能搜索

5. A\* 算法

我们简单的抽象一下，如果用  $f(M)$  表示：从起点S 终点E（经过M点）的距离，那它就可以表示成为两段距离之和，即：S→M的距离加上M→E的距离。如果我们用符号公式表示的话，就可以写成： $f(M) = g(M) + h(M)$ ，如图3-27所示：

S→M的距离+M→E的距离

3.3 启发式搜索

第三章 智能搜索

5. A\* 算法

我们扩展到M点的时候，S和M的距离就已经知道，所以 $g(M)$ 是已知的，但是M到E的距离 $h(M)$ 我们还不知道。常见的距离计算公式有这么几种：

- 1、曼哈顿距离：上面我们讲的横向格子数加纵向格子数；
- 2、欧式距离：两点间的直线距离 $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

除了上述的距离计算公式以外，还有一些变种的距离计算公式，如：对角线距离等等。这个就在具体的问题中做具体的优化了。

3.3 启发式搜索

第三章 智能搜索

A\*算法：回到布加勒斯特问题

到布加勒斯特的直线距离	
Arad	366
Bucharest	0
Cluj	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	226
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

3.3 启发式搜索

第三章 智能搜索

A\*算法：回到布加勒斯特问题

3.3 启发式搜索

第三章 智能搜索

A\*算法的应用

- 游戏AI
- 导航路径
- 资源分配
- 机器人运动
- 语言分析

3.3 启发式搜索

第三章 智能搜索

A\*算法

回忆：如何描述状态、动作、转换模型？  
启发式代价函数如何定？  
 $h(x)$  = 错位棋子个数  
 $g(x)$  = 移动步数

3.3 启发式搜索

第三章 智能搜索

A\* 算法

• 对比：

- 一致代价搜索
- A\* (hx基于错位)
- A\* (hx基于曼哈顿距离)

	4步	8步	12步
一致代价搜索	112	6,300	$3.6 \times 10^6$
基于错位的A*	13	39	227
基于曼哈顿距离的A*	12	25	73

——数据源：Andrew Moore

• 为什么？  
• 启发函数如果与问题求解越接近越好

3.3 启发式搜索

第三章 智能搜索

A\* 算法

- 启发式的设计方式是整个问题最关键的地方，选择哪一种刻画方式至关重要。
- 从问题条件出发：松弛问题
- 从结果出发：子问题
- 无关问题：经验和知识

3.3 启发式搜索

第三章 智能搜索

A\* 算法:从松弛问题出发设计可采纳的启发式

- 松弛问题：减少了行动限制的问题
- 如果问题定义是用形式语言描述的，那么有可能来自动构造它的松弛问题。例如，如果八数码问题的行动描述如下：
  - 棋子可以从方格A移动到方格B，如果A与B水平或垂直相邻而且B是空的。
- 通过去掉一个或两个条件，可以生成三个松弛问题：
  - 棋子可以从A移动到方格B，如果A和B相邻
    - 可以得出曼哈顿距离
  - 棋子可以从方格A移动到方格B，如果B是空的
    - 可以得出Gaschnig启发式
  - 棋子可以从方格A移动到方格B
    - 可以得出不在位的棋子数

3.3 启发式搜索

第三章 智能搜索

A\*算法: 3.6.3 从子问题出发设计可采纳的启发式

• 可以通过考虑给定问题的子问题的解来构造，比如上图我们只考虑将1234四个棋子移动到正确的位置上，不考虑其它棋子的位置。

3.3 启发式搜索

第三章 智能搜索

**A\*算法：从经验中学习启发式**

- 启发函数 $h(n)$ 用来估计从节点 $n$ 开始的解的代价。可以考虑从经验里学习。
- “经验”指的是求解大量的八数码问题，每个八数码问题的最优解都可以作为 $h(n)$ 学习的素材，从这些例子中，一个学习算法可以构造出 $h(n)$ 。
- 如果状态描述外还能刻画给定状态的特征，还可以用学习方法来做。
- 例如记“不在位的棋子数”为 $x_1(n)$ ，棋子距离它最终位置的距离为 $x_2(n)$ ，那么可以使用线性组合的方式构造预测结果：
$$h(n) = c_1x_1(n) + c_2x_2(n)$$

3.3 启发式搜索

第三章 智能搜索

$f(n) = g(n) + h(n)$ 讨论

- $f(n)=g(n)$ ,  $h(n)=0$ ,  $g(n)$ 为实际路径代价
  - 等价搜索 -- 效率不高，能得到最优解。
- $f(n)=h(n)$ ,  $g(n)=0$ ,
  - 爬山搜索 -- 效率极高，往往陷入局部最优。
- $f(n)=g(n)+h(n)$ ,  $g(n)>0$ ,  $h(n)>0$ ,
  - 最好优先搜索 -- 但是设计估价函数时要充分考虑 $g(n)$ 和 $h(n)$ 的数量级要相当。否则当 $g(n)>>h(n)$ 时，接近等价搜索； $h(n)>>g(n)$ 时，接近贪婪搜索。
- $f(n)=g(n)+h(n)$ ,  $g(n)>0$ , 且 $g(n) \geq g^*(n)$ ,  $h(n) \leq h^*(n)$ 
  - A\*算法 -- 为了兼顾搜索效率，在满足上述条件前提下，要尽可能取较大的 $h(n)$ 值，尽可能提高搜索效率。
- $f(n)=0$ , 即 $g(n)=0$ ,  $h(n)=0$ ,
  - 盲目搜索。

3.3 启发式搜索

第三章 智能搜索

**A-star算法举例**



3.3 启发式搜索

第三章 智能搜索

**课堂实验（设计报告）**

请各位同学使用A4纸进行提交：

- 学号，姓名
- 设计报告的内容

传教士和野人问题（The Missionaries and Cannibals Problem）：在河的左岸有 3 个传教士、3 个野人、1 条船。传教士想用这条船将所有成员都运过河，但是收到以下条件约束：

- 船每次最多只能装运两个；
- 在任何岸边野人数目都不能超过传教士，否则传教士会遭遇危险。
- 船每次不能空载。

请基于课程中关于状态空间图与搜索方法，尝试规划出一种确保所有成员安全过河的计划。

解决问题思路包括：

- 状态该如何表示？初始状态、结束状态分别是什么？状态空间有多少种，其中合理状态空间有哪些？
- 动作集合包括什么？
- 设计一种启发式算法（写出算法流程图）进行该问题求解，并说明该启发式算法是否是 A\*算法？

额外加分项目：

- 假设有  $M$  个传教士、 $N$  个野人，船每次最多只能装运  $K$  个。将上述启发式算法泛化到该问题求解。
- 编写程序，输入  $M$ 、 $N$  和  $K$ ，程序返回从初始状态到结束状态的运送方法。

感谢聆听

