成　绩	
评卷人	

研究生	冯朗
学　号	2015363065

武漢紡織大學

研 究 生 课 程 论 文

论文题目	启发式搜索技术
完成时间	2020. 10. 25
课程名称	人工智能
专　　业	电子信息
年　　级	2020

武汉纺织大学研究生处制

注：（1）正文宋体小四 1.5 倍行距；（2）不要列出源程序；（3）页数不少于 8 页；

一、 引言

关于搜索理论的研究是人工智能的核心领域，在人工智能领域，搜索即对某种问题的求解过程，而搜索方式就是求解该问题的方式或者模型。从提出问题（即初始状态）到解决问题（即目标状态），整个的求解过程， 事实上就是一个状态空间搜索[1]的过程。常用的状态空间搜索的算法有宽度优先和深度优先，但是深度优先和广度优先作为盲目搜索（无信息的搜索）算法，存在很多缺陷，效率较差。由于无信息的搜素算法缺陷比较大，针对许多领域提出了有信息搜索算法，启发式搜索（Heuristic Search）。

本文旨在介绍启发式搜索技术中贪心算法和 A*算法的基本思想，以及对其存在的一些问题进行讨论，并对此提出一些想法与思考，以供日后的学习和参考。

本文主要有以下几个部分组成，首先会介绍贪心算法和 A*算法，以及对算法的问题及其中的关键技术做一个讨论，然后用一个实例来进行验证和分析，接着讨论一些相关的应用和发展，最后做一个总结展望。

二、 相关工作

1. 启发式搜索

启发式搜索，又称为有信息搜索，与无信息的搜索算法相比较，能够更加有效的求解问题，对问题的效率有很大的提升，原因在于，该搜索方式会优先扩展代价最低的节点，代价最低的节点通常是最优的扩展目标。

此类算法会引入一个代价函数，与一致代价搜索（UCS）类似,它们都需要对节点的代价对节点的优先级进行排序，从而选择优先扩展哪些节点。

因此，我们需要引入一个代价函数或者说评价函数（evaluation function）$f(n)$。它给出我们对某一个节点 n 的代价，从而决定访问 n 的优先级。通常会优先探索$f(n)$小的节点，这个探索的算法和一致代价搜索非常像。区别在于一致代价搜索是根据已经探索过的路径信息，即总是扩展路劲

[1] 状态空间搜索就是将问题求解过程表现为从初始状态到目标状态寻找这个路径的过程。由于求解问题的过程中分枝有很多，主要是求解过程中求解条件的不确定性，不完备性造成的，使得求解的路径很多这就构成了一个图，我们说这个图就是状态空间。问题的求解实际上就是在这个图中找到一条路径可以从开始到结果。这个寻找的过程就是状态空间搜索。

消耗最小的节点,来给出接下来探索的意向。而启发式搜索则根据额外的信息，也就是我们这里所讨论的代价函数。

如果我们定义从初始节点到节点 n 的总和的路劲代价（已探索过的路劲信息）$g(n)$为,则一致代价搜索就相当于$f(n) = g(n)$。

此时添加一个启发函数[2]（Heuristic Function）$h(n)$表示节点 n 到目标节点的代价估计值，则有：1、$h(n) \geq 0$；2、$h(n)$越小越接近目标节点；3、$h(n) == 0$,则 n 达到目标节点。最佳优先搜索的代价函数可定义为：

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

可以看出，当令$h(n) \equiv 0$时，此时的$f(n) = g(n)$,即代价函数就是一致代价搜索，也就是说一致代价函数可以说是最佳优先搜索的一个特例。

同理，如果令$f(n) = g(n) = depth(n)$,则该算法相当于宽度优先搜索，由此可以看出$f(n)$的变化对算法影响很大。

2. 贪心搜索

若令$f(n)$中$g(n) \equiv 0$，即$f(n) = h(n)$则该搜索是贪心最佳优先搜索。贪心算法的基本思想是，总是试图优先选择距离目标更近的节点，即$h(n)$更小的节点。总体来说，它总是只看当前的情况，做出目前状态下最优的选择，在贪心搜索中，它并不从整体上考虑最优的路径，他只从当前的状况选择下一步最优的选择，也就是说它更像是一种局部的最优解，而不是全局的最优解。

由于贪心算法的搜索结果是一种局部最优解，因此，它甚至可能在全局上产生最坏的搜索结果。在不记录已经访问节点的情况下，贪心搜索连完备性都不具有，即在问题有解的情况下贪心算法可能找不到解或者陷入死循环。

只有在具备最佳子结构[3]的情况下，此时贪心算法才能找到最优解。通

[2] 启发函数 h(n)是一个估算函数,估算节点 n 到目标节点的代价,一般如果一个搜索算法使用了启发函数，那么我们认为该算法是启发式搜索算法（有信息的搜索算法），否则，称它为无信息的搜索算法。

[3] 假设“问题”是“替代方案”的集合，并且让每个替代方案都有关联的成本 c（a）。我们的任务是找到一组最小化 c（a）的替代方案。假设可以将备选方案划分为子集，即每个备选方案仅属于一个子集。假设每个子集都有自己的成本函数。可以找到每个成本函数的最小值，也可以找到全局成本函数的最小值（限于相同子集）。如果这些最小值与每个子集都匹配，那么很明显，不能从一组备选方案中选取全局最小值，而只能从我们定义的较小的局部成本函数的最小值组成的组中选取。如果最小化局部函数是“较低阶”的问题，并且（特别是）在有限数量的减少之后，该问题变得微不足道，则该问题具有最佳子结构。

常，如果可以通过归纳证明贪心算法在每个步骤都是最佳的，则可以使用贪心算法解决具有最佳子结构的问题。

3. A*搜索

3.1 A*的启发函数一些问题

A* 搜索算法是最佳优先搜索算法中的代表算法，它既考虑了实际代价函数$g(n)$，也考虑了启发函数$h(n)$,即$f(n)=g(n)+h(n)$。

A* 算法的基本思想是，由于$g(n)$表示开始节点到节点 n 的实际路径代价，而$h(n)$表示节点 n 到目标节点的最小路径代价的估计值，那么$f(n)$就表示从开始节点到目标节点且经过节点 n 的最小代价解的估计代价，因此，如果需要找到从开始节点到目标节点的最小代价解，那么首先扩展 $f(n)=g(n)+h(n)$值最小的节点就是合理有效的。

为什么 A*算法能够找到最优解呢？这与启发函数$h(n)$有很大的关系，前文已提到启发函数$h(n)$对启发式搜索的影响，所以启发函数如果设计的有问题，那么 A*算法的效果可能并不理想，或者说是失败的。

为了确保 A*算法能够搜索到最优的解，对于启发函数$h(n)$必须添加两个限制：

1. $h(n)$必须是可接受的或具有可接受性。可接受性指$h(n)$肯定不会过高的估计节点 n 到目标节点代价，因为$h(n)$是一个估计值，那么可接受性意味着$h(n)$小于等于节点 n 到目标节点的实际最小代价。通俗来讲，可以理解为两点之间直线最短。如果我们做一个假设，$h(n)$大于节点 n 到目标节点的实际代价，那么存在一条路径使 n 到目标节点的代价比$h(n)$小，则$h(n)$并不是最优的代价函数。所以此时认为$h(n)$函数的设计失败的。
2. $h(n)$必须是一致的或具有一致性的或具有单调性。一致性指对于每个节点 n 及其后继节点n'满足 n 到目标节点的代价小于 n 到n'的代价加上n'到目标节点的代价，用公式可以表示为

$$h(n) \leq cost(n,n') + h(n')$$

其中，$cost(n,n')$表示 n 到n'的单步代价（后继节点一步就可以到达）。通俗来说，可以理解为三角形两边之和大于第三边。满足一致性一

定满足可接受性，反之不一定。

3.2 A*算法的完备性问题

为了确保 A*的完备性,对状态空间做出两个弱假设:（绝大多数搜索任务都满足）

1. 每个节点的后继节点个数是有限的；

2. 状态空间的单步代价是非零代价（大于某个非常小的数值 a>0）

如果 A*算法不能找到目标节点，那么意味着有无限多个节点 n,满足$f(n) = g(n) + h(n) \leq f^*$,此处$f^*$表示最优解的最小代价。此时，节点个数无限多，与第一条相悖；由于单步单价大于 0，每次探索都会使得$f(n)$增大，总会达到$f(n) > f^*$的时候，故不等式不成立。

综上所述，在后继节点有限且单步代价大于 0 时，A*算法是完备的。

4. 启发函数

4.1 启发函数对 A*算法的影响

启发函数$h(n)$与 A*算法息息相关，是 A*算法的关键所在。

对于启发函数，有一种极端情况，如果$h(n)$是 0，则只有$g(n)$起作用，此时 A*算法演变成一致代价搜索算法，这保证能找到最短路径。

如果$h(n)$经常都比从 n 移动到目标的实际代价小（或者相等），则 A*保证能找到一条最短路径。$h(n)$越小，A*扩展的结点越多，运行就得越慢。

如果$h(n)$精确地等于从 n 移动到目标的代价,则 A*将会仅仅寻找最佳路径而不扩展别的任何结点，这会运行得非常快。尽管这不可能在所有情况下发生，你仍可以在一些特殊情况下让它们精确地相等(估计代价等于实际代价)。只要提供完美的信息，A*会运行得很完美，认识这一点很好。

如果$h(n)$有时比从 n 移动到目标的实际代价高,则 A*不能保证找到一条最短路径，但它运行得更快。

另一种极端情况，如果$h(n)$比$g(n)$大很多，则只有$h(n)$起作用(近似$g(n) \equiv 0$)，A*算法演变成 BFS 算法。

所以我们得到一个很有趣的情况，那就是我们可以决定我们想要从 A*

中获得什么。理想情况下，我们想最快地得到最短路径。如果我们的目标太低，我们仍会得到最短路径，不过速度变慢了；如果我们的目标太高，那我们就放弃了最短路径，但 A*算法运行得更快。因此，关于 A*算法中启发函数的设计非常的重要。

A*的这个特性非常有用。例如，你会发现在某些情况下，你希望得到一条好的路径，而不是一条完美的路径，不管是最求速度还是效果，都与启发函数的设计有关。

4.2 启发函数的构造原则

代价函数的构造没有什么规律可循,原则上只要有利于问题的求解,可以随意定义。但是根据前文分析，我们可以得出结论：启发函数$h(n)$必须可采纳,能够解决问题。否则否则 A*算法就只能称作 A 算法[4]。

$h(n)$可采纳就一定能找到最短路径,但$h(n)$与实际值$h^*(n)$不能差得太远。差得越远， A*算法最后的搜索拓扑就越接近一个完全的宽度优先搜索。理论上来说，$h(n)$=$h^*(n)$是最好的,估计值就是实际值,但这在实际中是不可能达到的。

所以，就要提到启发函数$h(n)$的信息量问题,所谓的信息量,就是在估计一个结点的值时的约束条件，如果信息越多(约束条件越多),则启发函数越准,能够排除的结点越多,那么 A*算法性能越好。但是，$h(n)$的信息越多,计算量就越大,耗费的时间或者内存也就越多，此时应该适当的减小$h(n)$的信息量，即约束条件，来提高算法的效率。,所以设计$h(n)$时需要根据具体应用环境来进行综合平衡设计。

三、 算法描述（伪代码）和验证（结果与分析）

1．最佳优先搜索算法

最佳优先搜索算法是各类算法的一般形式，下面给出最佳优先搜索算法的伪代码：

Input:

[4] A 算法不能保证最后的结果是最优的,但它的速度可能是非常快。在某些 RPG 游戏中，实时性要求高,也不一定非要得到最优解。

```
    A graph G and a vertex v of G
    came_from is a DICT, initialized with {}
Output:
    return: GOAL or None
    came_from is changed
def BestFS(G, v, came_from):
    frontier = PriorityQueue()
    cost_so_far = {}
    frontier.enqueue(v, 0)
    cost_so_far[v] = 0
    came_from[v] = None
    while not frontier.is_empty():
        v = frontier.dequeue()
        if v is a goal:
            return v
        else:
            for all edges from v to w in G.adjacentEdges(v):
                new_cost = cost_so_far[v] + G.cost(v, w)  #新的代价 =
起点到 v 节点的代价 + v 节点到 w 节点的代价
                if w not in cost_so_far or new_cost < cost_so_far[w]:
                    cost_so_far[w] = new_cost
                    priority = new_cost + heuristics(goal, w)     #
新的优先级优先级=新代价+启发函数h(w)的代价
                    frontier.put(w, priority)          #更新优先级
                    came_from[w] = v
    return None
```

这里给出的最佳优先搜索的算法，其中加入了启发函数$h(n)$。

2．八数码问题

八数码问题游戏，如图 3-1 所示，包括一个 3x3 的棋盘，棋盘上有 8 个数

字棋子和一个空格。与空格相邻的棋子可以滑动到空格中。 游戏目标是要达到一个特定的状态，如图中右侧所给出的状态。该问题可形式化如下：

1.状态： 状态描述指明 8 个棋子以及空格在棋盘 9 个方格上的分布。

2.初始状态： 任何状态都可能是初始状态。注意要到达任何一个给定的目标，可能的初始状态中恰好只有一半可以作为开始。

3.后继函数：用来产生通过四个行动（上下左右） 能够达到的合法状态。

4.目标测试： 用来检测状态是否能匹配图 3-1 中所示的目标布局（其他目标布局也是可能的）。

5.路径耗散：每一步的耗散值为 1，因此整个路径的耗散值是路径中的步数。

我们对 8 数码问题的一些部分进行抽象，行动被抽象为它的起始和结果状态，忽略了当棋子滑动时所走过的中间过程（即不考虑物理过程）。我们保留和游戏规则有关的描述，避免陷入所有物理操作的细节。八数码问题共有 9!/2= 181 440 个可达到的状态，并且很容易求解。15 数码问题（在 4x4 的棋盘上） 有大约 1.3 万亿个状态，用最好的搜索算法求解一个随机的实例的最优解需要几毫秒。 24 数码问题（在 5X5 的棋盘上） 的状态数可达 1025 个，求解随机实例的最优解可能需要几个小时。

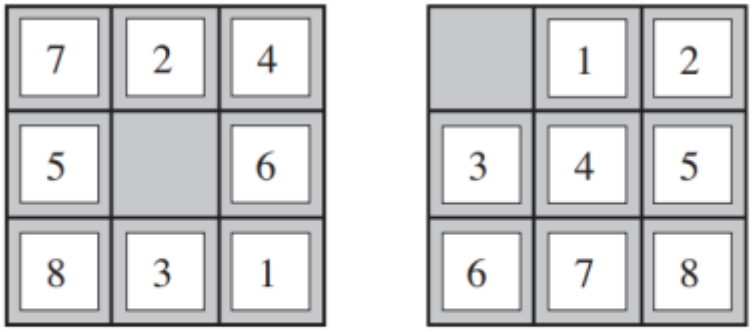


图 3-1

2.1 逆序数

在一组序列 P 中有 n 个元素，若对 i<j，有$a_i > a_j$，则称元素a_j是元素a_i的一个逆序，元素a_i的逆序数记为$R(a_i)$。整个序列 P 的逆序数记为$R(P)$。

$$R(P) = \sum_{i=0}^{n} R(a_i)$$

若将序列 P 的元素都放入数组 S[1 - n*n]中，则求逆序数的算法如下（伪代码）：

```
def inversions(P)
    t = 0
    i = 1
    while i <= n*n:
        if S[i] not is empty:
            for j in range(i+1,n*n):
                if S[j] > S[i]:
                    t = t + 1
        i = i+1
    R(P) = t
    return R(P)
```

2.2 可解性判断

8 数码问题不一定是可解的，8 数码问题有解的条件是：初始状态和目标状态的逆序数奇偶性相同[6]。例如，在图 3-1 中，Start State 的状态可表示为序列{7,2,4,5,0,6,8,3,1}，Goal State 可表示为序列{0,1,2,3,4,5,6,7,8}。在不考虑 0，即将 0 看做空格时，初始状态逆序数为 6+1+2+2+2+2+1=16，目标状态逆序数为 0。所以该 8 数码问题有解。

[6] 8 数码谜题扩展到任意 n 阶时，有可解性判断定理:1. 对奇数阶棋盘，只有当初始状态序列与目标状态序列二者的逆序数奇偶性相同时问题有解。2. 对偶数阶棋盘，只有当初始状态序列与目标状态序列二者各自的逆序数加其空格所在行数所得结果的奇偶性相同时问题有解。

3．验证分析

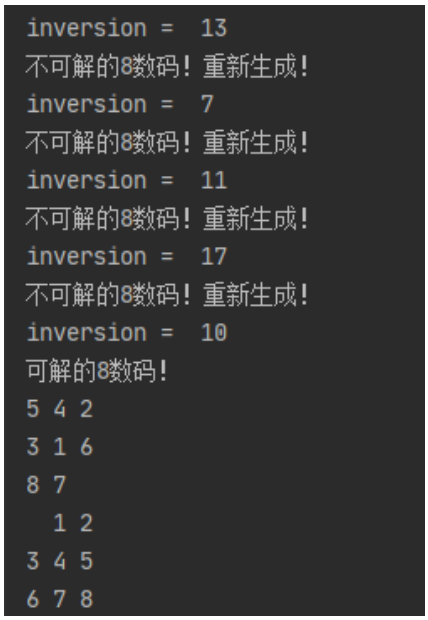


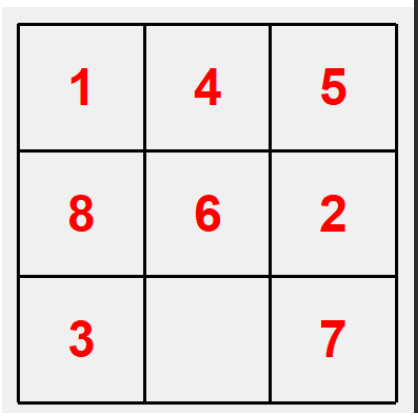
图 3-2

在图 3-2 中，能够看出生成的 8 数码问题解决受到奇偶性的影响，当产生的初始的逆序数为奇数时，生成的 8 数码问题是不可解的，只能重新随机生成 8 数码问题的初始状态。

4．启发函数(伪代码)

```
def _calcHeuristics(self):
    heuristics = 0
    for row in range(Puzzle8.HEIGHT):
        for col in range(Puzzle8.WIDTH):
            item = self[row, col]          #row 为行数，col 为数
            if item != ' ':
                index = Puzzle8.ITEMS.index(item)   #index 为
当前位置上的数码值
                row1 = index // Puzzle8.WIDTH    #移动行数
                col1 = index % Puzzle8.WIDTH #移动列数
```

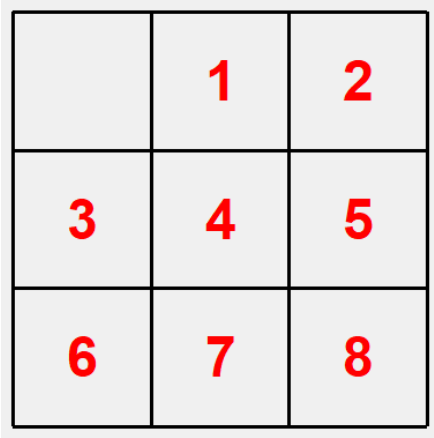
```
                heuristics += abs(row1-row) + abs(col1-col)
#启发函数 h(x）为 目标从当前状态移动到目标状态需要走的格子数
        return heuristics
```




```
row = 0,    row1 =0,
col = 0,    col1 = 1,
index = 1, item = 1,      h(item) = 1
row = 0,    row1 =1,
col = 1,    col1 = 1,
index = 4, item = 4,      h(item) = 2
row = 0,    row1 =1,
col = 2,    col1 = 2,
index = 5, item = 5,      h(item) = 3
row = 1,    row1 =2,
col = 0,    col1 = 2,
index = 8, item = 8,      h(item) = 6
row = 1,    row1 =2,
col = 1,    col1 = 0,
index = 6, item = 6,      h(item) = 8
row = 1,    row1 =0,
col = 2,    col1 = 2,
index = 2, item = 2,      h(item) = 9
row = 2,    row1 =1,
col = 0,    col1 = 0,
index = 3, item = 3,      h(item) = 10
row = 2,    row1 =2,
col = 2,    col1 = 1,
index = 7, item = 7,      h(item) = 11
heuristics= 11
```

Start State　　初始位置数据

图 3-3




```
row = 0,    row1 =0,
col = 0,    col1 = 1,
index = 1, item = 1,      h(item) = 1
row = 0,    row1 =1,
col = 1,    col1 = 1,
index = 4, item = 4,      h(item) = 2
row = 0,    row1 =0,
col = 2,    col1 = 2,
index = 2, item = 2,      h(item) = 2
row = 1,    row1 =1,
col = 0,    col1 = 0,
index = 3, item = 3,      h(item) = 2
row = 1,    row1 =1,
col = 2,    col1 = 2,
index = 5, item = 5,      h(item) = 2
row = 2,    row1 =2,
col = 0,    col1 = 0,
index = 6, item = 6,      h(item) = 2
row = 2,    row1 =2,
col = 1,    col1 = 1,
index = 7, item = 7,      h(item) = 2
row = 2,    row1 =2,
col = 2,    col1 = 2,
index = 8, item = 8,      h(item) = 2
heuristics= 2
```

Goal State　　目标位置数据

图 3-4

该启发函数$h(n)$的设计是由初始状态到目标状态下移动的格子数，以图 3-3 为例，左图中，数码 1 在（0,0）位置上，数码 1 移动到目标位置只需要向下移

动一格，所以$h(1)$=1,在位置数据中，我们只参考首位数值的数据和最后的heuristics 总值,因为$h(n)$为一个累加值,后续的数值需要对前面的$h(n)$值减 1，例如，在初始位置图中，数码 4 在（0,1）位置上，移动到目标位置（1,1），只需要移动一格，然而输出的 h(4)=2，这是由于累加了数码 1 移动到目标位置的数值为 h(1)=1。

然后看目标状态中，h(n)的数值基本没有什么变化，因为接近目标状态时所需要的移动的数码就变少了。而且最终的 h(n)总值为 2,意味着只需要移动两步。比初始状态中第一次移动的 h(n)总值 11 要低的多。

四、 算法应用

启发式搜索被广泛的应用于各个领域之中。它作为一种有信息的搜索策略，对很多的问题都能得到高效的解决。

1. 无人机航迹规划

基于无人机导航系统的自身特点,无人机在导航过程中会出现无法精确定位的情况,从而产生定位误差。如果不能及时校正随时间累积的定位误差,会使无人机无法到达预定目的地,从而导致飞行任务失败。为避免这种情况的发生,研究了考虑定位误差的无人机航迹快速规划问题。以航迹距离最短为目标,考虑定位误差校正约束与航迹约束,用启发式深度优先搜索+回溯算法来求解问题,并在此算法基础上加入模拟退火机制对解的质量进行优化。以某飞行区域的数据为例进行仿真实验,结果表明启发式深度优先搜索+回溯算法可以快速有效地求解考虑定位误差的无人机航迹规划问题。

2. 机器人路径规划

面对三维空间移动机器人从起始点到终止点的最短路径问题,边缘点树启发式搜索(Tree-EP)算法,该方法将地图空间进行密度可调的三维离散化处理,根据障碍安全距离筛选出障碍物的可靠边缘点信息,再利用树扩散架构选出最能引导搜索方向的潜力点进行扩散搜索,最终得出最短路径。

3. 社交信任路径研究

在社交媒体中,信任传递在用户交互关系的建立上发挥着至关重要的作用.实际应用中,通常将信任传递过程应用于推荐系统来预测起始用户对特

定目标用户的信任程度,从而更好地作出下一步决策。选择的信任路径是否较优与预测的准确性息息相关。针对路径长度和信任值在整条路径上的值分布,提出一种新的加权启发式搜索信任预测模型.该模型将改进的经典 A*算法应用于信任网络进行路径寻找,其中,改进的 A*算法在寻路过程中使用了二次启发并将启发函数设置为筛选条件进行路径筛选。该模型最终得到的信任路径具有相对较好的鲁棒性,相对提高了预测的准确性,而且在信任累加计算中融入了信任的衰减。

4. 气象服务

当今,随着移动互联、大数据搜索技术的快速发展,气象服务发展面临着服务市场的开放和信息技术的快速发展两大机遇与挑战,现代电力、交通、旅游、能源等多个行业用户及公众对气象服务产品提出更加个性、更精细的需求。原有气象服务客户在服务产品升级时,如何在最大化效率的同时,最小化投入显得尤为重要。因此,利用启发式搜索可以通过指导搜索向最有希望的方向前进,用最小代价进行新气象服务产品研发并能提供个性化的服务成为气象服务客户的迫切需求。

五、 结论与展望

因为自己算法掌握得不是很熟练，在算法的理解及验证（读代码）方面花费了大量的时间，所以本文并没有做过多深入的拓展，更多的是根据讲义内容， 简要了解了 A* 算法及 启发式函数的相关问题，并结合相关资料做了基本实现及验证，同时，通过了解一些具体的应用，加深了对 A*算法和启发函数的认识和学习，在学习的过程中，我也认识到，该算法不仅在传统领域应用广泛，在目前比较火热的机器学习，人工智能领域也有着不可获缺的作用。 因此，本文在扎实自己算法功底的基础上，希望能给自己以后机器学习提供一些创新思路，为以后的学习做一些参考。

六、 参考文献

[1] StuartJ.Russell, PeterNorvig, 诺维格,等. 人工智能:一种现代的方法[J]. 2013.

[2] 杜小勤.《人工智能》 课程系列, Part Ⅲ:启发式搜索技术,2018/10/08

[3] Wikipedia:Optimal_substructure .. Accessed on 20 August 2020

[4]钟敏.A 算法估价函数的特性分析[J].武汉工程职业技术学院学报, 2006, (2) .

[5]崔振兴,顾治华.基于 A*算法的游戏地图最短路径搜索[J].软件导刊,2007(17):145-147.

[6] 冯晓辉,马光思.数码谜题求解的算法设计及其扩展研究[J].计算机技术与发展,2009,19(08):110-112+116.

[7] 高双槐.普适数码问题及其求解方法[J].中国科技信息,2008(20):46+49.

[8] 龙振海,林泓.8 数码问题求解算法的改进与实现[J].中国高新技术企业,2010(03):19-21.

[9] 马光思.组合数学[M].西安:西安电子科技大学出版社, 2002.

[10] 宋文, 伊良忠,牟行军.15-谜问题的可达性判定[J].电子科技大学学报, 2004, 33 (5) :604-607.

[11] 吴玉文,牛智越,韩倩倩.基于启发式搜索算法的无人机航迹快速规划[J].科学技术与工程,2020,20(20):8394-8399.

[12] 胡晓敏,梁天毅,王明丰, 等.新型树启发式搜索算法的机器人路径规划[J].计算机工程与应用,2020,56(11):164-171.

[13] 魏桐,童向荣.基于加权启发式搜索的鲁棒性信任路径生成[J].南京大学学报（自然科学版）,2018,54(6):1161-1170.

[14] 李亚玲,郭洁,詹万志.基于启发式搜索的气象服务产品功能配置研究[J].高原山地气象研究,2019,39(1):91-95.