南開大學
Nankai University

《人工智能》实验报告

(2023~2024 **学年第一学期**)

实验名称：基于八数码问题的搜索策略对比

学　　院：软件学院

姓　　名：陈高楠

学　　号：2112966

指导老师：张玉志

2023 年 11 月 26 日

目录

基于八数码问题的搜索策略对比

1 问题描述

人工智能中的搜索问题是指找到从初始状态到目标状态的路径的问题。这些问题通常涉及到在复杂、有时是动态变化的环境中寻找解决方案。八数码问题是人工智能搜索中的一个典型问题，目前解决八数码的主要方法有深度优先搜索 (DFS)、宽度优先搜索 (BFS)、启发式A * 算法等等。本文将对这几种算法解决八数码问题的效率进行探讨，并进行比较，从步数、时间、结点数等各参数，通过具体的实验数据分析，进一步验证各算法的特性。

2 问题分析

2.1 问题建模

八数码涉及一个 3x3 的网格，里面带有 8 个数字的方块和一个空块。每次搜索需要对方块进行移动，来逐步改变棋盘格局以搜索到目标状态。对八数码问题进行建模，棋盘如下面三图所示，图一为搜索的初始状态，图三为搜索的目标状态，也就是需要达到的状态。图二为搜索的过程。可以把八数码的过程抽象成一个搜索的过程，用树来表达。

2	5	4
3		6
1	7	8

图 1: 初始状态

2	3	
1	4	5
7	8	6

图 2: 中间状态

1	2	3
4	5	6
7	8	

图 3: 目标状态

2.2 八数码解的存在性

八数码并不一定都有解，它的解的存在性与初始状态逆序有关。在一个排列中，如果一对数的前后位置和大小顺序相反，即前面的数大于后面的数，那么它们就成为一个逆序；排列中逆序的总数就成为这个排列的逆序数。而可以证明：如果初始状态的逆序与目标状态的

逆序奇偶性相同，则该初始状态可以搜索到目标状态。如图 1 的序列为 {2，5，4，3，0，6，1，7，8}，逆序为 1+3+2+1+0+1+0+0+0=8，而目标序列为 {1，2，3，4，5，6，7，8，0}，逆序为 0，二者逆序奇偶性相同，因此图 1 的状态有解。

3 算法设计与分析

3.1 盲目搜索

如果在搜索过程中没有利用任何与问题有关的知识或者信息，则称为盲目搜索。盲目搜索主要有深度优先搜索，广度优先搜索，深度优先迭代搜索算法等。下面将对这几种盲目搜索进行探讨。

3.1.1 DFS 深度优先搜索算法

深度优先搜索算法 (Depth-First-Search，DFS) 是一种用于遍历或搜索树或图的算法。它的思路是每次选择一个深度最深的节点进行扩展，如果有相同深度的多个节点，则按照事先规定选择，尽可能深的搜索树的分支。如果该节点没有子节点了，就回溯到除了该节点外最深的节点进行扩展，直到找到问题的答案，或者再也没有节点可以拓展，则表示问题无解。深度优先不一定能找到最优解，而是找到可行的解。

深度优先算法解决八数码问题的步骤如下：

(1) 建立 open 表和 close 表，用于存储未访问的状态和已经访问的状态。将初始状态加入到 open 表中，开始搜索。

(2) 判断 open 表是否为空，如果为空则失败。如果不为空，将 open 表中最后一个状态加入到 close 表中，并计算是否到达最大深度，如果没到则继续运行。如果到了则强制回溯，重复执行第二步。

(3) 对于该节点，拓展它的子节点加入到 open 表的前端。

(4) 判断 open 表的表头是否为目标状态，如果是则成功，如果不是则回到第二步继续运行，一直到返回失败或者成功。成功则回溯找到路径输出。

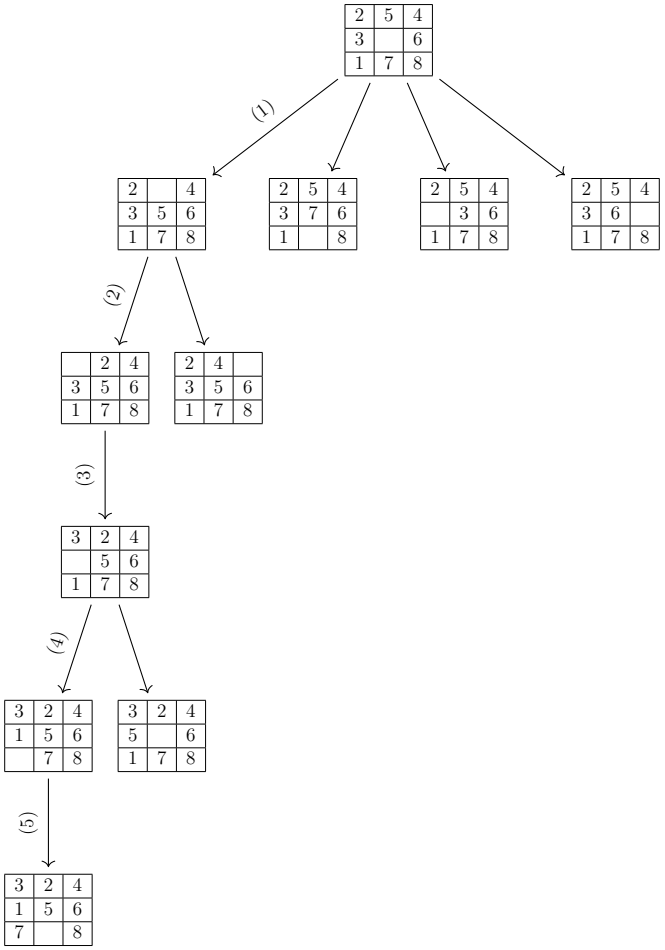


图 4: 深度优先示例

如图所示，图中省略已经走过的路径，沿着树的深度遍历树的节点，尽可能深的搜索树的分支，当节点 v 所在的边都已经被搜索过就开始回溯。这里需要注意的是，对于很多问题，这样可能会导致沿着错误的方向越走越远，为了防止这样的事情发生，应当给深度设置一个临界值，到达临界值则停止并强制进行回溯，以防止搜索深度过深无法获得答案。

3.1.2 IDDFS **深度优先迭代搜索算法**

迭代深度优先搜索（Iterative Deepening Depth-First Search，简称 IDDFS）是一种结合了深度优先搜索（DFS）和广度优先搜索（BFS）特性的搜索算法。它结合了广度优先和深度优先，通过逐渐增加搜索深度的方式进行搜索，每次限制深度优先搜索的深度，从而搜索找到最短路径。深度优先迭代搜索也能找到最优解。迭代深度优先搜索算法解决八数码问题的步骤如下：

(1) 初始设限制深度为 0，建立 open 表和 close 表，用于存储未访问的状态和已经访问

的状态。将初始状态加入到 open 表中，开始搜索。

(2) 对于当前限制深度执行 DFS，当超过限制深度时则回溯，一直到 open 表为空或者找出目标状态。

(3) 如果 open 表为空仍然找不到结果，则将深度限制加 1，重复执行 (1)(2) 步。

(4) 一直到找出目标状态。成功则回溯找到路径输出。

IDDFS 深度优先迭代搜索算法如下所示：

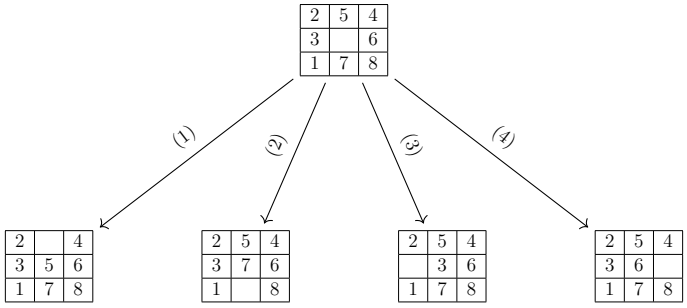


图 5: 深度为 1 时

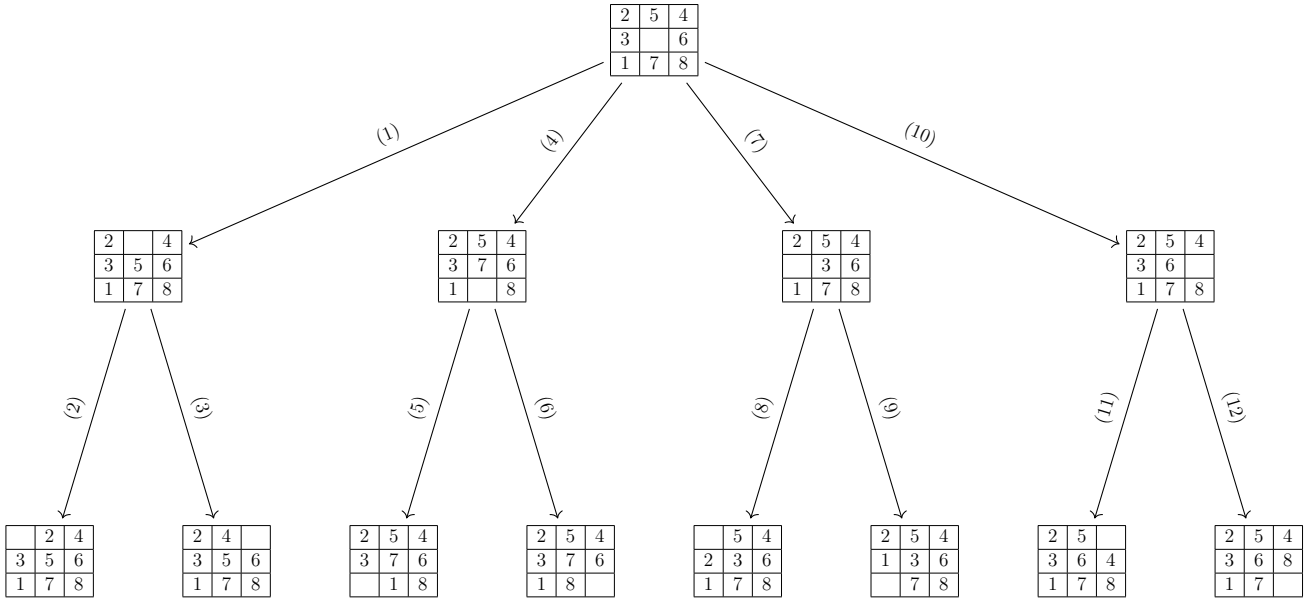


图 6: 深度为 2 时

如图所示，当当前深度限制找不到答案时，则将深度限制 +1 并进行重复搜索，值得注意的是，深度优先迭代搜索在新的迭代中，前一迭代中已经访问过的节点会被重新访问，导致计算上的冗余。

3.1.3 BFS 广度优先搜索算法

广度优先搜索（Breadth-First Search, BFS）是一种用于遍历或搜索树或图的算法。与深度优先策略相反，广度优先搜索优先搜索深度浅的节点，每次选择深度最浅的叶节点进行拓展，如果深度相同则按照事先约定的从深度最浅的几个节点中选择一个。与深度优先不同，在问题有解的情况下，广度优先搜索一定可以找到最优解。

广度优先搜索解决八数码问题的步骤如下：

(1) 建立 open 表和 close 表，用于存储未访问的状态和已经访问的状态。将初始状态加入到 open 表中，开始搜索。

(2) 判断 open 表是否为空，如果为空则失败。如果不为空，将 open 表中第一个加入到 close 表中。

(3) 对于该节点，拓展它的子节点加入到 open 表的前端。

(4) 判断 open 表的表头是否为目标状态，如果是则成功，如果不是则回到第二步继续运行，一直到返回失败或者成功。成功则回溯找到路径输出。

BFS 广度优先搜索示例如下图所示：

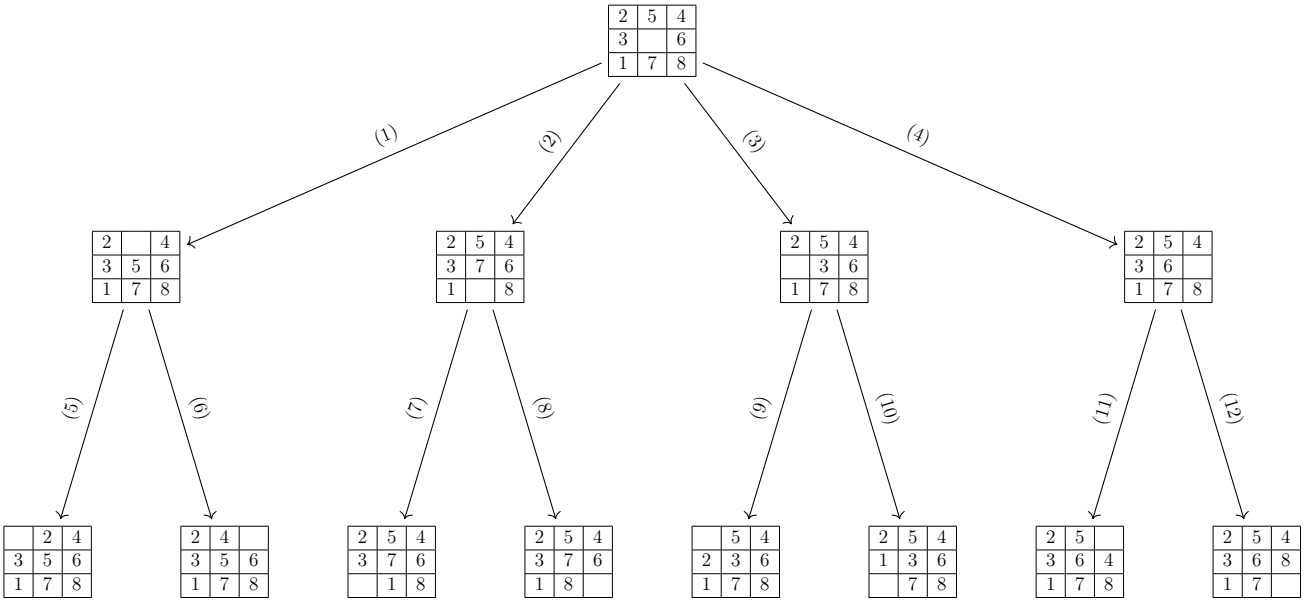


图 7: 广度优先搜索

如图所示，图中省略已经走过的路径，沿着树的广度遍历树的节点，尽可能广的搜索树的分支，由于 BFS 需要保存访问过的路径，因此需要占用较大的搜索空间，会随着搜索深度的增加成几何级数增加。

3.2 启发式搜索算法

前面介绍了盲目搜索算法，搜索范围比较大，所以效率比较低。如果在搜索过程中引入启发信息，减少范围，从而尽快找到解，则称为启发式搜索。启发式搜索算法是一种在搜索过程中使用启发式方法来优化搜索效率的算法。它们广泛应用于问题解决中，尤其是在问题空间庞大、无法穷举所有可能性的情况下。这类算法的核心在于使用一个启发式函数（也称为评估函数）来估计从当前状态到目标状态的最佳路径。如何评估呢？可以给出评估函数：

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

其中 n 是待评价的节点，g(n) 是初始节点 s 到节点 n 的最佳路径耗散值的估计值，h(n) 是节点 n 到目标节点 t 的最佳路径耗散值的估计值，也就是启发函数。所以 f(n) 为初始节点 s 到目标节点 t 的最佳路径耗散值的估计值。在八数码中，g(n) 已经确定为该节点的深度值，而启发函数可以根据问题定义给出不同的函数。

如何定义好的启发函数成为使用评估函数的关键所在。当启发函数取 0 时，此时评估函数 f(n) = g(n), 此时的搜索过程只考虑从起点到当前节点的实际成本，而不考虑从当前节点到目标的预估成本。这种情况下，搜索算法会优先扩展累积成本最小的节点，因此称为最佳优先搜索，其目的是找到成本最低的路径。如果 f(n) = h(n), 这时搜索过程完全忽略了从起点到当前节点的实际路径成本，只根据对目标距离的启发式估计来选择节点，这种算法被称为贪婪最佳优先搜索。这种方法可能会更快地找到目标，但不一定能找到成本最低的路径。

尽管启发式算法的评估函数有很多种，但它的搜索逻辑都是一样的。启发式搜索的步骤如下：

(1) 建立 open 表和 close 表，用于存储未访问的状态和已经访问的状态。将初始状态加入到 open 表中，开始搜索。

(2) 判断 open 表是否为空，如果为空则失败。如果不为空则继续下一步。

(3) 对于该节点，拓展它的子节点。如果该子节点不在 open 表也不在 close 表，则直接加入到 open 列表的后端。然后根据评估函数对 open 表进行排序。如果在 open 表，且估值函数值优于 open 队列中对应的子节点，则将 open 队列中的子节点替换成该子节点，然后根据评估函数对 open 表进行排序。如果在在 closed 表中，且估值函数值优于 closed 表中对应的子节点，则将 closed 表中对应的子节点移除，并将该子节点加入 open 优先队列。这一步的目的是倾向于找到评估函数最小也就是更接近于正确答案的路径。

(4) 判断 open 表的表头是否为目标状态，如果是则成功，如果不是则回到第二步继续运

行，一直到返回失败或者成功。成功则回溯找到路径输出。

3.2.1 贪婪最佳优先搜索算法

贪婪最佳优先搜索 (Greedy Best-First Search) 是一种启发式算法，它的代价函数 $f(n) = h(n)$, 它只关注于对目标距离的启发式估计来选择节点，而完全忽略已经走过的代价。这种算法的特点是它在每一步都尝试朝着似乎最接近目标的方向前进，因此搜索十分迅速就能找到可行解。但是这种贪心的性质也意味着它可能会忽略较长的初始路径，而这些路径可能最终会导致更低的总成本，因为当前代价小，以后不一定代价小，有可能会存在找不到解的情况。同时，这种迅速通常是以牺牲整体最优解为代价的，所以它并不一定能得到最优解。在八数码中，如果贪婪最佳优先搜索算法能够得到解，那么它的搜索速率往往是优于其他搜索算法的。

3.2.2 最佳优先搜索算法

最佳优先搜索算法 (Best-First Search Algorithm) 是一种启发式算法，可以将它看做广度优先搜索算法的一种改进。它的代价函数 $f(n) = g(n)$, 在八数码问题中，g(n) 可以简单地表示为到达当前状态的步数或深度。在八数码中，当启发式函数 f(n) 只考虑 g(n)，也就是从起始状态到当前状态的代价（如步数或深度），而不考虑到达目标状态的潜在代价，尽管它是一种启发式算法，但在这种特定情况下，它的效率和广度优先搜索相似，因为它们都是按照节点的发现顺序进行搜索，而不是根据节点离目标的近似距离。

3.2.3 A* 算法

启发式算法没有对评估函数进行任何规定，因此得到的结果如何也不好评定。如果启发式函数满足如下条件：

$$h(n) \leq h^*(n)$$

则可以证明如果问题有解时，算法一定可以得到耗散值最小的结果，也就是最佳的解。该启发式算法被称为 A* 算法。其中 $h^*(n)$ 是从状态 n 到目标状态的实际最短路径长度。在八数码中，$h^*(n)$ 可以用曼哈顿距离表示。因为在八数码中，每一方块每一次只能移动一格，而它最少需要移动它到目标方块位置的曼哈顿距离才能达到目标状态。因此曼哈顿距离提供了到达目标状态所需步数的下界估计。有了 $h^*(n)$ 函数，我们便可以给出四种可采纳的 $h(n)$ 启发函数。

(一) 基于曼哈顿距离

曼哈顿距离是指在格点上，从一个点到另一个点的路径长度，只允许水平或垂直移动。在八数码中，每一方块每一次只能移动一格，而且每次移动只能减少该方块的曼哈顿距离最多一单位，曼哈顿距离提供了到达目标状态所需步数的下界估计。而它最少需要移动它到目标方块位置的曼哈顿距离才能达到目标状态，因此它绝对不会夸大损失，满足 $h_{manhattan}(n) \leq h^*(n)$ 的条件，所以基于曼哈顿距离的启发函数满足条件，是可采纳的启发函数。

(二) 基于欧几里得距离

欧几里得距离是两点之间的“直线”距离。由三角形的两条边加起来大于第三边的定理可知，当在八数码问题求解时，某个方块到目标方块的欧几里得距离小于等于曼哈顿距离，而基于曼哈顿距离的启发函数是满足条件的，因此 $h_{euclidean}(n) \leq h_{manhattan}(n) \leq h^*(n)$ 成立，所以基于欧几里得距离的启发函数也满足条件，是可采纳的启发函数。

(三) 基于切比雪夫距离

切比雪夫距离即为 $d_(p,q) = \max_i(|p_i - q_i|)$，也就是点距离目标在任何一个坐标轴上的最大差值。在八数码中，切比雪夫距离即某个不在目标位置的方块要达到与目标位置同行或同列的最短步骤，切比雪夫距离恒小于等于曼哈顿距离，因为曼哈顿距离是点距离目标的每个坐标轴差值之和，因此曼哈顿距离大于等于切比雪夫距离。$h_{chebyshev}(n) \leq h_{manhattan}(n) \leq h^*(n)$，所以基于切比雪夫距离的启发函数也满足条件，是可采纳的启发函数。

(四) 基于不在位的方块数

当某一方块不在该有的位置时，它至少需要移动一次才能到达目标的位置。而某不在原有位置的方块距离目标位置的曼哈顿距离 $d \geq 1$，因此基于不在位的方块数的启发函数满足条件，是可采纳的启发函数。

还有很多符合条件的启发函数，也可以把最佳优先搜索算法看作是启发函数为 0 的 A* 算法，因此最佳优先搜索算法得到的结果也一定是最短路径。

下面是 A* 算法的图示。

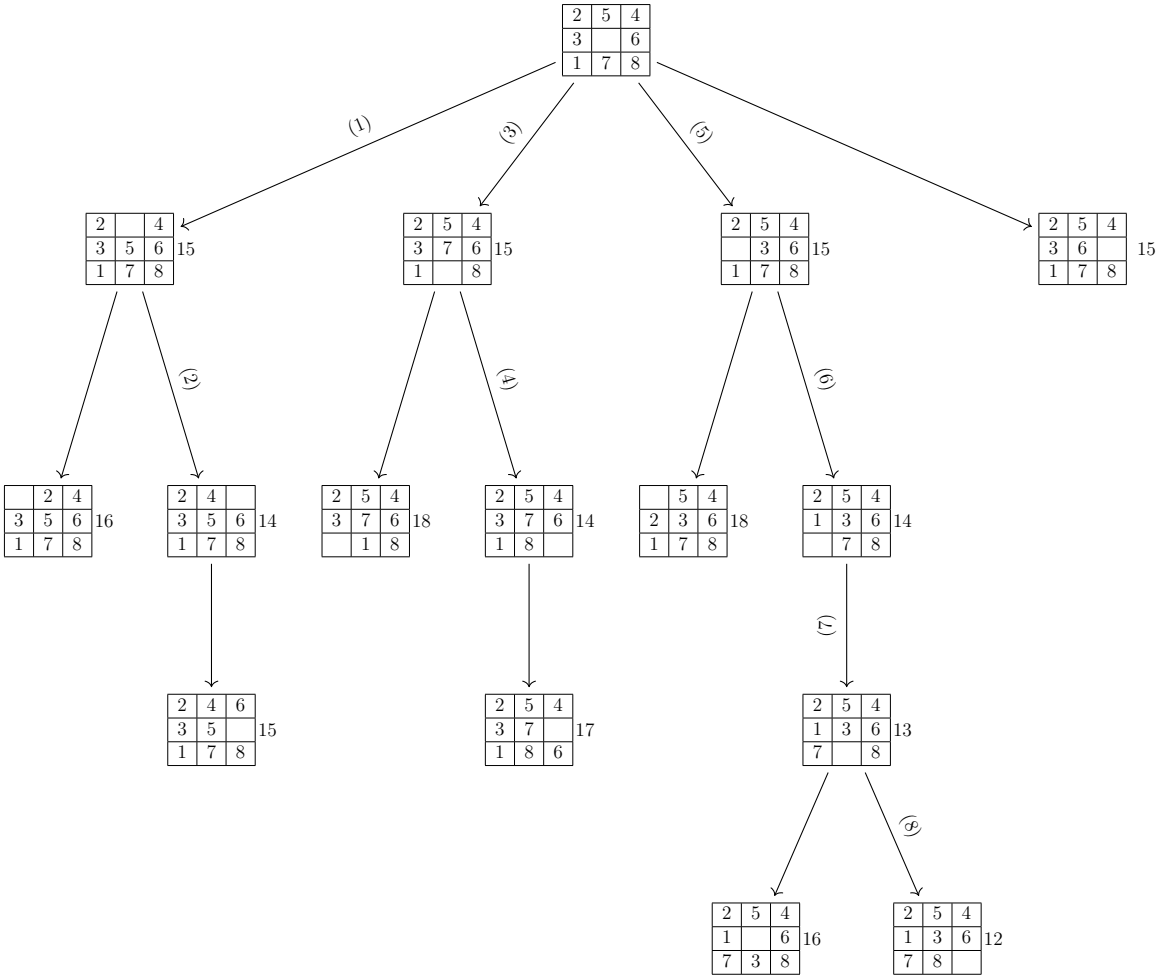


图 8: A* 搜索

如图所示，启发函数采取基于曼哈顿距离的启发函数，节点旁边的数字即为 f(n), 线上数字代表顺序。先从初始节点开始向下延伸并计算 f(n), 每次都从最小的 f(n) 所在的节点开始搜索。

3.3 八数码最长链的求取

八数码一共九个方块，不同的排列顺序一共有 9! = 362,880 种，可以通过 python 的生成器来遍历 362880 种初始状态，并对其进行最佳路径的搜索，记录路径长度最长的路径。当然，并不是所有的初始状态都存在解，需要通过初始状态的逆序和目标状态逆序的奇偶性来进行判定。经过测试，求解八数码问题的最佳路径中最长的路径为 31，有 2 种初始状态，分别如下图。这说明在八数码问题中，任何一个有解的初始状态，它到目标状态的最优路径必定不会超过 31 步。

8	6	7
2	5	4
3		1

图 9: 八数码最难问题 1

6	4	7
8	5	
3	2	1

图 10: 八数码最难问题 2

4 结果统计与分析

本次实验共测试 12 种不同的八数码初始状态 (目标状态不包括在内)，分别对应不同的最佳探索路径。实验采用五种算法，分别是 BFS 算法,DFS 算法,IDDFS 算法, 贪婪最佳优先搜索算法，最佳优先搜索算法，还有 A* 算法 (四种不同的代价函数)。DFS 限制深度为 100。因为测试算法结果的时间受计算机系统影响很大, 因此本次实验时间平均测试 5 次取平均值。测试的数据如下:

表 1: 测试的数据

次序	最佳步数	具体例子
1	0	[[1, 2, 3], [4, 5, 6], [7, 8, 0]]
2	2	[[1, 2, 3], [4, 5, 6], [0, 7, 8]]
3	4	[[0, 1, 3], [4, 2, 6], [7, 5, 8]]
4	6	[[4, 1, 2], [5, 0, 3], [7, 8, 6]]
5	8	[[1, 2, 3], [7, 0, 4], [5, 8, 6]]
6	10	[[4, 5, 1], [2, 0, 3], [7, 8, 6]]
7	12	[[2, 7, 3], [1, 0, 4], [8, 6, 5]]
8	14	[[5, 1, 6], [4, 3, 8], [0, 2, 7]]
9	16	[[5, 1, 6], [4, 0, 8], [2, 3, 7]]
10	18	[[5, 6, 0], [4, 1, 8], [2, 3, 7]]
11	20	[[0, 1, 2], [3, 5, 4], [7, 6, 8]]
12	22	[[0, 1, 2], [3, 4, 7], [5, 6, 8]]
13	24	[[0, 1, 2], [3, 4, 7], [6, 8, 5]]

4.1 运行时间分析

表 2: 搜索时间比较 (s)

最佳步数	DFS	BFS	IDDFS	Greedy	Best	A*
2	5.6100e-05	0.0001	3.5300e-05	7.6399e-05	0.0001	6.40e-05
4	0.0031	0.0004	0.0001	8.640e-05	0.0005	8.37e-05
6	0.0106	0.0019	0.0005	0.0001	0.0025	0.0002
8	0.0017	0.0067	0.0001	0.0001	0.0125	0.0004
10	0.0073	0.0133	0.0008	0.0003	0.0545	0.0008
12	12.5652	0.0400	0.0414	0.0062	0.2138	0.0007
14	4.2315	0.0961	0.0851	0.0084	0.7357	0.0007
16	1.0218	0.7693	0.2320	0.0086	9.5706	0.0018
18	17.1084	5.0145	1.3477	0.0092	36.4498	0.0022
20	1.0760	24.5726	0.7298	0.0048	267.4102	0.0504
22	1.6074	109.7784	4.2787	0.1303	904.53	0.2007
24	1.3660	1872.6471	14.3699	0.0128	1466.63	0.8510

从表中可以看出，盲目搜索所消耗的时间在某些时候远大于启发式搜索，且具有一定的随机性，这是因为盲目搜索具有一定的随机性，如果盲目搜索刚好进入了目标状态所在的路径，那么他的搜索时间消耗就很少，且比启发式搜索少了排序的一步。而如果没有进入，盲目搜索所消耗的时间则会大量超过启发式搜索，甚至会出现超时的情况。在搜索最优解为 24 的路径时，使用 BFS 花了 1872 秒才搜索出最佳结果。

在启发式搜索中，最佳优先搜索算法效果是最差的，消耗的时间比盲目搜索还多。这是因为 f(n)=g(n) 时，它并没有起到一个很好的启发作用。而贪心算法在搜索的过程中效率是最高的，这是因为贪心算法忽略了现有的代价而只考虑对目标距离的启发式估计，因此他会以很快的速度朝着目标状态前进。但是由于不考虑当前的代价，它可能会因为进入一条没有用的路径而消耗大量时间，同时，贪心算法得到的结果也不一定是最优解。A* 算法是相对稳定的，效率也是相对最高的，得到的也是最优解。

4.2 遍历节点数目比较

表 3: 遍历节点数目

最佳步数	DFS	BFS	IDDFS	Greedy	Best	A*
2	3	7	7	3	7	3
4	233	25	41	5	25	5
6	1087	108	203	11	107	11
8	108	371	504	9	335	17
10	715	771	2432	19	632	39
12	1071101	2129	7666	159	1550	30
14	389912	4447	18125	253	2929	34
16	96178	21017	65306	255	10734	64
18	437154	55083	129652	258	22659	90
20	111540	176070	346247	84	49480	744
22	115332	477317	1210156	563	84751	1531
24	100582	1433813	3631467	315	130204	3449

从表中可得，盲目搜索所需要遍历的节点数目会出现比较庞大的情况，这是因为如果盲目搜索的时候进入一条无关的路径，会花费大量的时间来探索无用的节点。

在启发式搜索中，最佳优先搜索算法遍历的节点数目也比较庞大，因为它的代价函数并不能带来很好的启发效果。而贪心算法如果有解，它所遍历的节点数目往往是最少的，因为它的思路是尽可能接近目标状态，因此它会以最快的方式接近目标状态。而 A* 算法因为考虑了当前的代价和当前到目标状态的启发式代价，因此探索的节点会比贪心算法多，但远小于盲目搜索。

4.3 解路径长度分析

表 4: 解路径长度 (s)

最佳步数	DFS	BFS	IDDFS	Greedy	Best	A*
2	2	2	2	2	2	2
4	98	4	4	4	4	4
6	94	6	6	6	6	6
8	98	8	8	8	8	8
10	100	10	10	10	10	10
12	96	12	12	28	12	12
14	98	14	14	42	14	14
16	100	16	16	42	16	16
18	100	18	18	44	18	18
20	100	20	20	20	20	20
22	100	22	22	54	22	22
24	100	24	24	50	24	24

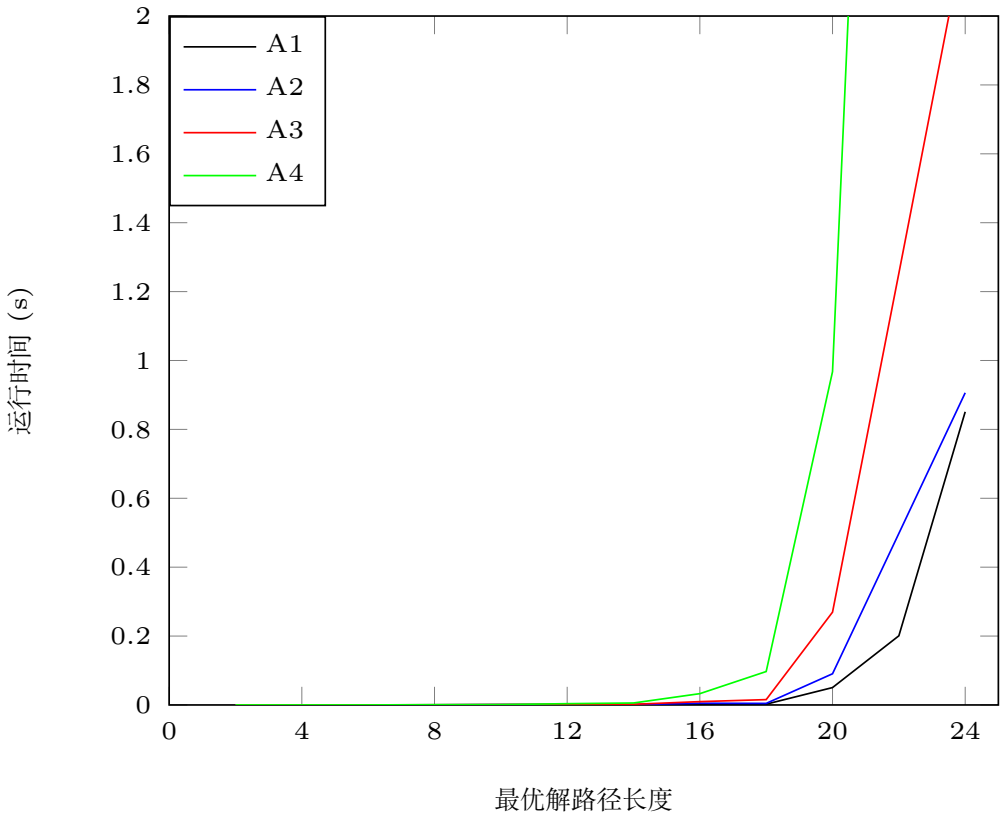
从表中可以看出，DFS 和贪心算法得到的往往不是最优解，因为他们更倾向于找出一条可行路径而不是最佳路径，所以他们的结果不一定是最佳路径但一定是可行性路径。而 BFS，IDDFS，最佳优先搜索算法和 A* 算法更倾向于找到最佳的路径，因此得到的解便是最佳路径。

4.4 A* 不同启发式算法分析

A* 是效果最好的搜索算法，但是不同 h(n) 函数的效果也不同。下面比较的是 A* 不同启发式算法的表现，其中 A1 代表基于曼哈顿距离的启发函数，A2 代表基于欧几里得距离的启发函数，A3 代表基于切比雪夫的启发函数，A4 代表基于不在位方块数的启发函数。

表 5: 搜索时间比较 (s)

最佳步数	A1	A2	A3	A4
2	6.40e-05	4.89e-05	4.64e-05	4.13e-05
4	8.37e-05	8.86e-05	8.60e-05	7.82e-05
6	0.000238	0.000165	0.000165	0.000124
8	0.000443	0.000368	0.000460	0.000738
10	0.000874	0.000817	0.000961	0.001056
12	0.000730	0.000927	0.001735	0.003707
14	0.000711	0.000946	0.001907	0.005717
16	0.001895	0.005085	0.009666	0.032817
18	0.002282	0.004508	0.015487	0.097052
20	0.050467	0.090529	0.269370	0.967549
22	0.200773	0.497972	1.253161	5.332105
24	0.851029	0.906228	2.243781	27.333721



尽管使用 A* 算法能够极大加速搜索的过程且能得到最优解，但使用不同启发函数效果也是不一样的。从图中可以看出，搜索效率：A1>A2>A3>A4。

表 6: 搜索节点数比较 (s)

最佳步数	A1	A2	A3	A4
2	3	3	3	3
4	5	5	5	5
6	11	9	9	7
8	17	17	21	27
10	39	37	42	46
12	30	34	58	121
14	34	41	79	186
16	64	116	237	613
18	90	152	373	1149
20	744	1048	1885	3721
22	1591	2542	3949	8492
24	3449	3355	5390	18856

搜索节点数 A4>A3>A2>A1，说明使用基于不在位方块数来进行 A* 搜索会搜索出更多的冗余节点。搜索效率和搜索的节点数与他们能否更逼近最少代价 $h^*(n)$ 是相关的，可以看到在八数码，逼近下界 $h^*(n)$ 的函数中，逼近的排名：曼哈顿距离 > 欧几里得距离 > 切比雪夫距离 > 不在位方块数，刚好与搜索的效率相对应。同时也可以看到，比较搜索的次数最少的排序也是这个顺序。这是因为使用曼哈顿距离的启发函数更符合八数码的游戏规则，而使用欧几里得距离，切比雪夫距离，不在位方块数为启发函数很有可能低估了实际的移动步数，从而导致搜索效率不是最优的。

4.5 DFS **不同深度限制结果分析**

通过改变 DFS 的限制深度观察解的变化，

表 7: 解的路径长度比较 (s)

最佳步数	深度 25	深度 50	深度 75	深度 100
2	2	2	2	2
4	20	50	74	98
6	24	48	72	94
8	16	46	68	98
10	22	50	74	100
12	24	48	74	96
14	22	48	72	98
16	24	50	72	100
18	24	48	74	100
20	24	50	72	100
22	24	50	74	100
24	24	50	74	100

虽然深度不同得出的解的长度也不同，但是他们都很接近深度限制，这也充分反映了深度优先搜索的特点，即优先向深度深的节点进行搜索，找出一条可行路径，因此 DFS 也很难得到最优解。

5 实验结论及心得体会

通过研究，可以得到如下结论：

①在盲目式搜索算法中，BFS 和 IDDFS 算法一定能够得到最优解，DFS 不一定能得到最优解，但是 DFS 在某些时候能较快得出一条可信路径，节点冗余相对较少，而 BFS 和 IDDFS 虽然能够得出最佳路径，但是节点冗余较多，对内存影响比较大。

②IDDFS 的搜索效率比 BFS 略高，但是 IDDFS 搜索节点所占的内存比 BFS 要高很多，两者都能得到最优解。

③ 在启发式搜索算法中，贪心最佳优先搜索算法不一定能够得到最优解，但是它在某些时刻的搜索效率是最高的。而 A* 算法既能得到最优解，算法效率又比盲目式搜索高很多。最佳优先搜索算法的效率是最低的，它的评估函数并不能很好地起到启发的作用。

④ 在 A* 算法中，基于曼哈顿距离的启发函数搜索的表现是最好的，最差的是根据不在位的方块数得出的启发式函数，但是都能得到最优解。