八数码问题解法效率比较及改进研究

付宏杰[1],王雪莹[2],周 健[2],周孙静[2],朱 珠[2],张俊余[2]

(1. 吉林工程技术师范学院 信息工程学院,吉林 长春 130052;
2. 东北师范大学 信息与软件工程学院,吉林 长春 130117)

摘 要:八数码问题是人工智能中的一个典型问题,目前解决八数码问题的搜索求解策略主要有深度优先搜索、宽度优先搜索、启发式A*算法。对这些算法进行研究,重点对A*算法进行适当改进,使用曼哈顿距离对估价函数进行优化。对使用这些算法解决八数码问题的效率进行比较,从步数、时间、结点数、外显率等各参数,通过具体的实验数据分析,进一步验证各算法的特性。

关键词:八数码;深度优先搜索;宽度优先搜索;A*算法;曼哈顿距离

DOI:10.11907/rjdk.161867

中图分类号:TP312 **文献标识码**:A **文章编号**:1672-7800(2016)009-0041-05

1 问题描述

八数码问题就是在一个大小为3×3的九宫格上,放置8块编号为1~8的木块,九宫格中有一个空格,周围(上下左右)的木块可以和空格交换位置。对于问题,给定一个初始状态(如图1(a)初始状态),目标状态(如图1(b)目标状态)是期望达到1~8顺序排列的序列,并且空格在右下角,问题的实质就是寻找一个合法的移动序列。

2 问题分析和模型表示

2.1 模型建立

对于棋格,每个位置给定一个编号,从左上角开始顺序编号(见图2),对于任意的状态,记为p = s[0]s[1]s[2]s[3]s[4]s[5]s[6]s[7]s[8],图1初始状态中的状态可以表示为p=2 8 3 1 6 4 7 0 5,图2目标状态中的状态可以表示为p′ = 1 2 3 4 5 6 7 8 0。

验。实验结果发现,改进算法的推荐结果准确性与召回率提高10%左右。其中准确性和召回率的计算公式如公式(3)和公式(4)所示。

$$Precision@N = \frac{\sum_{u} |R(u,N) \cap T(u)|}{\sum_{u} R(u,N)} \quad (3)$$

$$Recall@N = \frac{\sum_{u} |R(u,N) \cap T(u)|}{\sum_{u} T(u)} \quad (4)$$

4 结语

本文针对基于位置的推荐算法在划分数据子集时产生的数据稀疏问题,提出了对不同节点的推荐列表进行线性加权的解决办法。实验结果表明,改进的推荐算法提高了推荐结果的准确性。

参考文献:

[1] 项亮. 推荐系统实践[M]. 北京:人民邮电出版社,2012.
[2] GOWALLA[EB/OL]. http://mashable.com/category/gowalla/.
[3] 刘树栋,孟祥武. 一种基于移动用户位置的网络推荐方法[J]. 软件导刊,2014,25(11):56-74.
[4] MAFENGWO[EB/OL]. http://www.mafengwo.cn/
[5] CHAKPABORTY B. Integrating awareness in user oriented route recommendation system[C]. The International Joint Conference on Neural Networks, New Jersey: IEEE Press, 2012. 1-5.
[6] 徐雅斌,孙晓晨. 位置社交网络的个性化位置推荐[J]. 北京邮电大学学报,2007,38(5):118-122.
[7] FOURSQUARE[EB/OL]. https://foursquare.com/
[8] PLACES. GOOGLE[EB/OL]. http://places.google.com/rate

(责任编辑:杜能钢)

作者简介:付宏杰(1973-),女,吉林公主岭人,博士,吉林工程技术师范学院信息工程学院副教授,研究方向为人工智能、群体智能、机器学习。

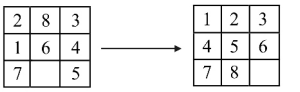

图 1 序列移动

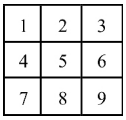

图 2 编号

2.2 分析解的存在性

不是每一个给定的初始状态都存在解，在分析之前，引入线性代数逆序和逆序数的概念：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。例如，排列 p=2 8 3 1 6 4 7 0 5 的逆序数为 1+6+1+0+2+0+1=11(不可解)。

使用线性代数理论可以论证，对于任意目标状态，只有初始状态的逆序数和目标状态的逆序数的奇偶性相同才有解(逆序数计算不包括 0 的逆序数)。例如 p'=1 2 3 4 5 6 7 0 8 的逆序数为 0，与 p 的逆序数奇偶性相同，因此是可解状态。

3 搜索策略

搜索算法本质上是一棵将初始节点作为根节点的四叉树。从初始节点开始，以空白格向 4 个方向的移动作为分支。求解的过程就是寻找一条从根节点到目标结点的路径的过程。

为了方便论述，取目标状态为：p=1、2、3、4、5、6、7、8、0，本文用上、右、下、左分别表示空格的向上、向右、向下、向左 4 种操作。

3.1 深度优先搜索算法(Depth-First-Search)

深度优先搜索策略是扩展当前结点，生成的子节点总是置于扩展栈的前端，这样搜索向纵深方向发展。

3.1.1 算法策略

①将起始结点 S 加入栈中，如果此结点是目标结点，则得到解；②如果栈为空，则无解，失败退出，否则继续执行步骤Ⅲ；③将栈顶元素(记作结点 N)从栈中取出；④如果结点 N 的深度等于最大深度，则转向步骤Ⅱ；⑤扩展结点 N 的所有结点，产生其全部的后继结点，并压入栈；⑥如果后继结点中有任一结点为目标结点，则求得解，否则转向步骤Ⅱ。

3.1.2 搜索图解

搜索图解如图 3 所示。

3.1.3 算法优缺点分析

深度优先算法在解决八数码问题时有一个致命缺点，就是必须设置一个深度界限，否则，搜索会一直沿着纵深方向发展，会一直无法搜索到解路径。即使加了限制条件，搜索到了解路径，解路径也不一定是最优解路径。

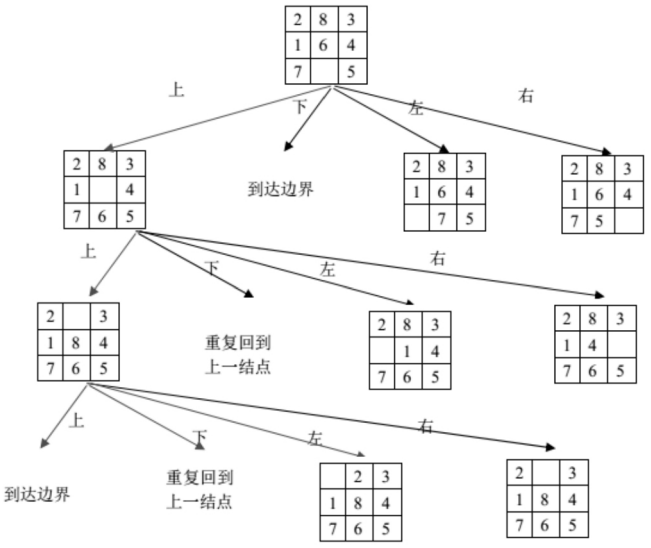

图 3 DFS 搜索图解

初始状态：5 6 8 4 0 1 2 3 7。运行结果图 4 所示。

```
time:1209ms
The cost was: 50.0
The number of nodes examined: 1167629
```

图 4 DFS 运行结果

缺点：如果目标节点不在搜索进入的分支上，而该分支又是一个无穷分支，就得不到解，因此该算法是不完备的。

优点：如果目标节点在搜索进入的分支上，则可以较快得到解。

3.2 宽度优先搜索算法(Breadth-First-Search)

宽度优先算法将扩展结点置于扩展队列的末端，使得搜索向横向发展。

3.2.1 算法策略

①将起始结点放入队列中，如果该起始结点为一目标结点，则得到解；②如果队列为空，则无解，失败退出，否则继续步骤Ⅲ；③将第一个结点(记作结点 N)放入队列中，并将它放入已扩展结点的队列中；④扩展结点 N，如果没有后继结点，则转向步骤Ⅱ；⑤将 N 的所有后继结点放到队列末端，并提供从这些后继结点回到结点 N 的指针；⑥如果 N 的任一后继结点是个目标结点，则找到一个解(反向追踪得到从目标结点到起始结点的路径)，成功退出，否则转向步骤Ⅱ。

宽度优先搜索对于八数码问题的解决情况，相较于深度优先搜索好，对于解决步数在 20～30 步的初始状态，可以在 300～500ms 内解决。

3.2.2 搜索图解

搜索图解如图 5 所示。

3.2.3 算法优缺点分析

宽度优先搜索，在搜索过程中，遵循一层一层往下搜索的搜索策略，它是一个完备的算法，找到的解一定是最优解。

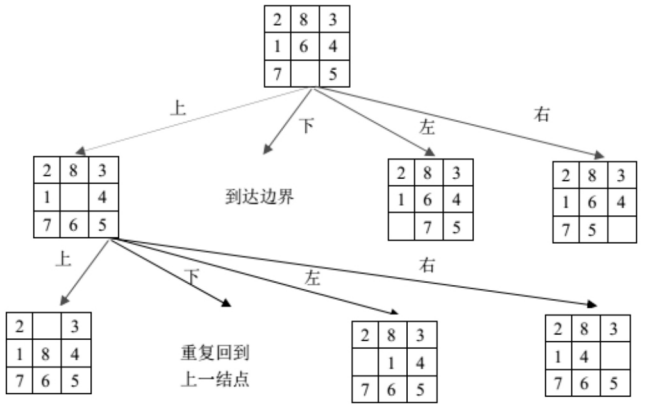

图 5　BFS 搜索图解

初始状态：5 6 8 4 0 1 2 3 7。运行结果图 6 所示。

```
time:2216ms
The cost was: 20.0
The number of nodes examined: 270672
```

图 6　BFS 运行结果

缺点：当目标节点距离初始节点较远时会产生许多无用的节点，搜索效率低，只能适用于到达目标结点步数较少的情况，如果步数超过 15 步，运行时间太长，则不再起作用。

优点：只要问题有解，则总可以得到解，而且是最短路径的解。

3.3　A＊算法

A＊算法的基本思想与广度优先算法相同，但是在扩展出一个结点后，要计算它的估价函数，并根据估价函数对待扩展的结点排序，从而保证每次扩展的结点都是估价函数最小的结点。

3.3.1　算法思想

A＊算法是一种常用的启发式搜索算法。在 A＊算法中，一个结点位置的好坏用估价函数来对它进行评估：

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

这里，$f'(n)$是估价函数，g(n)是起点到终点的最短路径值(也称为最小耗费或最小代价)，$h'(n)$是 n 到目标的最短路经的启发值。由于这个 $f'(n)$其实是无法预先知道的，因而实际上使用的是如下估价函数：

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

其中，$g(n)$是从初始结点到节点 n 的实际代价，$h(n)$是从结点 n 到目标结点的最佳路径的估计代价。在这里主要是 $h(n)$体现了搜索的启发信息，因为 $g(n)$是已知的。用 $f(n)$作为 $f'(n)$的近似，也即用 $g(n)$代替 $g'(n)$，$h(n)$代替 $h'(n)$。这样必须满足两个条件：① $g(n) \geqslant g'(n)$（大多数情况下都是满足的，可以不用考虑），且 f 必须保持单调递增；② h 必须小于等于实际的从当前节点到达目标节点的最小耗费 $h(n) \leqslant h'(n)$ 。第二点特别重要，可以证明应用这样的估价函数可以找到最短路径。

估价函数中，主要是计算 h，对于不同的问题，h 有不同的含义。这里的 h(n)代表的是当前结点状态与目标结点状态相比没有到位的棋子数，是最简单的估价函数。

3.3.2　算法策略

①建立一个队列，计算初始结点的估价函数 f，并将初始结点入队，设置队列头和尾指针；②取出队列头(队列头指针所指)的结点，如果该结点是目标结点，则输出路径，程序结束。否则对结点进行扩展；③检查扩展出的新结点是否与队列中的结点重复，若与不能再扩展的结点重复，则将它抛弃，若新结点与待扩展的结点重复，则比较两个结点的估价函数中 g 的大小，保留较小 g 值的结点。跳至Ⅴ；④如果扩展出的新结点与队列中的结点不重复，则按照其估价函数 f 的大小将它插入队列中的头结点之后待扩展结点的适当位置，使它们按从小到大的顺序排列，最后更新队列尾指针；⑤如果队列头的结点还可以扩展，直接返回步骤②，否则将队列头指针指向下一结点，再返回步骤②。

3.3.3　搜索图解

搜索图解如图 7 所示。

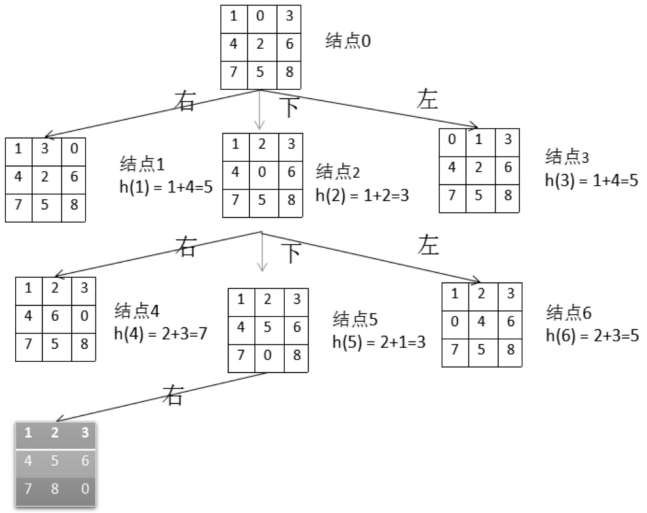

图 7　A＊算法搜索图解

3.3.4　算法优缺点分析

优点：A＊算法在绝大多数的情况下，在性能方面都远远优与 DFS 和 BFS。算法的主要运行性能，取决于估价函数 f 的选取。

缺点：由于算法本身的特点，因此根据估价函数找到的解路径不一定是最优解路径。

初始状态：5 6 8 4 0 1 2 3 7。运行结果图 8 所示。

```
time:381ms
The cost was: 26.0
The number of nodes examined: 17680
```

图 8　A＊算法运行结果

4　算法改进

4.1　状态表示方法压缩

有许多表示方法，比如一个 3＊3 的八数码盘可以压

缩成一个 int 值表示，但不适用于格子大于 9 的问题（如十五谜问题）。如果对空间要求很高，则还可以再压缩。本文采用整数（int）表示的方法。

表示方法如下：由于 int 的大小是 32 位（范围是[－2147483648－2147483648]），取一个 int 的低 9 位。为了方便寻找空格的位置，int 的个位用来放空格的位置（1－9）。而前 8 位，按照行从上到下，列从左到右的顺序依次记录对应位置上的数字。

4.2 哈希函数去重

高效避免访问同一个状态，最直接的方法是记录每一个状态访问与否，然后在衍生状态时不衍生那些已经访问的状态。基本思想是，给每个状态标记一个 flag，若该状态 flag = true 则不衍生，若为 false 则衍生并修改 flag 为 true。

每一次移动产生新状态时，先判断它是否已在两个链表中，若存在，则不衍生；若不存在，将其放入活链表。对于被衍生的那个状态，放入死链表。

为了记录每一个状态是否被访问过，需要有足够的空间。八数码问题一共有 9！这个数字并不是很大，采用哈希函数的方法，使复杂度减为 O(1)。

4.3 估价函数改进

通过找出不在位置的棋格个数形成的简单估价函数，不足以描述该结点到目标节点的路径长度。因此本文采用一种更为科学合理的距离函数，来描述两个状态结点之间的差距。设距离函数为 $h_2(n)$，其函数值为当前结点状态不在位置的棋格，到它目标结点需要移动的距离：

$$h_2(n) = |x - x_0| + |y - y_0|$$

使用此函数，对于 A* 算法的估价函数进行改进，能够取得比较明显的改进效果。

5 效率比较

5.1 概念

首先给出几个用来进行效率比价的变量：①步数（L）：从初始节点到达目标的路径长度；②时间（T）：搜索程序运行的时间，单位毫秒（ms）；③结点数（N）：整个过程中生成的节点总数（不包括初始节点）；④外显率（P）：搜索工程中，从初始节点向目标节点进行搜索的区域的宽度。

$$P = \frac{L}{N}; P \in (0,1]$$

5.2 实验数据分析

数据说明：①环境为 Windows 系统，语言为 Java，使用控制台命令输出算法时间；②目标状态 1 2 3 4 5 6 7 8 9 0；③由于运行时间受系统影响很大，具有一定的随机性，因而每个状态执行 3 次，取中位数作为最终结果时间；④“长度”为该初始状态对应的最短路径的长度。实验数据及结论如表 1～表 5、图 9～图 12 所示。

表 1 实验数据说明

序号	长度	初始状态（举例）
1	1	1 2 3 4 5 6 7 0 8
2	2	1 2 3 4 5 6 0 7 8
3	4	0 1 3 4 2 6 7 5 8
4	8	1 2 3 7 0 4 5 8 6
5	10	4 5 1 2 0 3 7 8 6
6	12	2 7 3 1 0 4 8 6 5
7	14	5 1 6 4 3 8 0 2 7
8	16	5 1 6 4 0 8 2 3 7
9	18	5 6 0 4 1 8 2 3 7
10	20	5 6 8 4 0 1 2 3 7

（1）八数码问题解路径长度比较。

表 2 实验数据分析

长度	DFS 路径长度	BFS 路径长度	A＊路径长度	改进算法路径长度
1	1	1	1	1
2	2	2	2	2
4	26	4	4	4
8	28	8	8	8
10	30	10	12	10
12	30	12	12	12
14	30	14	24	14
16	30	16	24	16
18	30	18	26	18
20	30	20	26	20

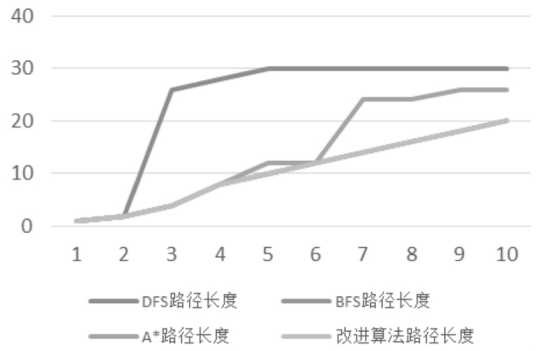

图 9 实验数据分析

（2）八数码问题解决时间比较。

表 3 实验数据分析

长度	DFS 时间/(ms)	BFS 时间/(ms)	A＊时间/(ms)	改进算法时间/ms
1	7	7	19	6
2	9	12	22	10
4	533	17	26	12
8	255	56	37	22
10	188	65	50	34
12	243	186	85	38
14	728	235	160	72
16	1198	311	334	142
18	3190	592	374	179
20	1038	2362	446	209

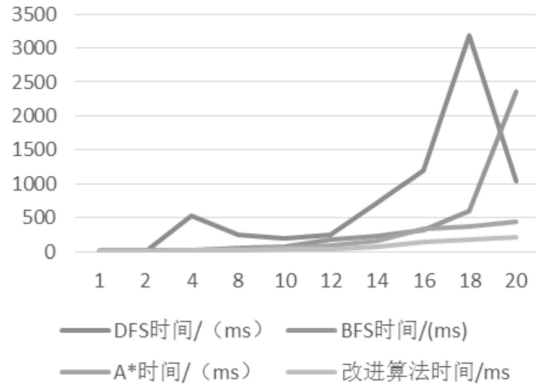

图 10　实验数据分析

(3)八数码问题结点数比较。

表 4　实验数据分析

长度	DFS 结点数	BFS 结点数	A* 结点数	改进算法 结点数
1	2	4	2	2
2	3	7	3	3
4	380475	27	5	5
8	5841	337	21	21
10	1491	998	56	46
12	4046	2843	242	188
14	5599209	4623	341	294
16	2473390	21013	6862	945
18	754267	47890	12725	1278
20	188553	270672	17680	4447

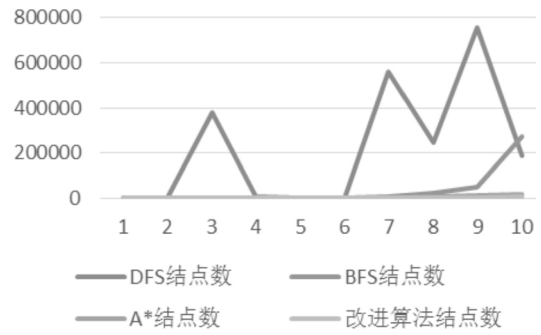

图 11　实验数据分析

(4)八数码问题外显率比较。

5.3　研究结论

通过研究，可得结论如下：①DFS 的解路径长度趋于搜索深度界限（本文界限是 30），搜索效率受深度影响很大，并且搜索结点冗余多、速度慢；②BFS 找到的一定是最优解，但是在算法效率上，虽然比 DFS 好，却远远不如A* 算法，同时BFS在搜索深度较深时，产生的冗余结点较多；③A* 算法在效率上相对最优，时间和空间上都比 DFS 和 BFS 更优，但缺点是，找到的解不一定是最优解；④改进的 A* 算法在时间复杂度和效率上都更优，空间利用率（外显率）与 A* 算法差距不大。总体而言，改进的算法取得了较好效果。

表 5　实验数据分析

长度	DFS 外显率	BFS 外显率	A* 结点数	改进算法 外显率
1	0.5	0.25	0.5	0.5
2	0.666 667	0.285 714 2	0.666 666 67	0.666 667
4	0.000 068	0.148 148 1	0.8	0.8
8	0.004 793	0.023 738 8	0.380 952 3	0.380 952 3
10	0.020 120	0.010 02	0.214 285 7	0.217 391 3
12	0.000 074	0.004 220 8	0.049 586 7	0.063 829 7
14	0.000 053	0.003 028 3	0.070 381 2	0.047 619
16	0.000 012	0.000 761 4	0.003 497 5	0.016 931 2
18	0.000 039	0.000 375 8	0.002 043 2	0.014 084 5
20	0.000 159	0.000 073	0.001 470 5	0.004 497 4

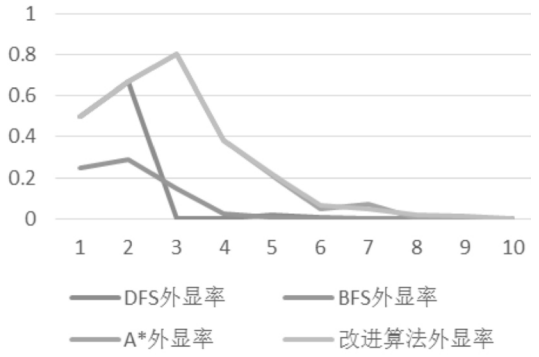

图 12　实验数据分析

参考文献：

[1] 蔡自兴. 徐光. 人工智能及其应用[M]. 北京：清华大学出版社，1996.

[2] 唐朝舜. 董玉德. 八数码的启发式搜索算法及实现[J]. 安徽职业技术学院学报，2004，3(3)：9-12.

[3] 陶阳. VS2008 环境下八数码问题的 BFS 算法设计与实现[J]. 电脑知识与技术，2009(26)：14-17.

[4] 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，2003，5(14)：14-18.

[5] 贺计文，宋承祥，刘弘. 基于遗传算法的八数码问题的设计及实现[J]. 计算机技术与发展，2010(3)：20-23.

[6] 姚全珠，赵双瑞. 基于状态空间搜索的 ETL 过程优化[J]. 计算机工程与应用，2007(26)：44-46.

（责任编辑：孙　娟）