N 码问题

1. 引言

N 码问题，或者 n^2-1 码问题，其包含 n^2-1 个方块（分别标号为 1 至 n^2-1），放置在 n*n 的棋盘中。问题的目标是通过移动方块使得棋盘上的方块和给定的目标顺序一致[1]。如图 1 所示为一个已经解决的 15 码问题。

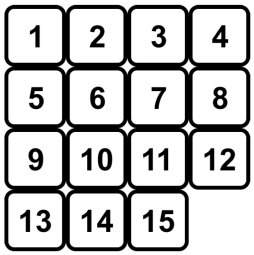

图 1. 一个已经解决的 15 码问题

15 码问题通常被认为是 Sam Loyd 发明的一种游戏，但是研究[2]表明 Sam Loyd 既没有发明 15 码游戏，也没有对其风靡产生促进作用。这个游戏的风靡始于 1880 年 4 月份的欧洲。Loyd 首次于 1891 年声称发明了这个问题。 而实际发明人是纽约州的 Noyes Chapman，他于 1880 年 3 月就为其申请了专利。

解决 N 码问题通常使用启发式搜索。当使用搜索算法时，通常会考虑包含已经展开的节点（存放在 expanded 列表中）和已生成但未扩展的节点（存放在 frontier 列表中）。搜索算法是以特定 f(n)函数值的顺序访问 frontier 列表中的节点并展开。最著名的启发式搜索算法是 A*，其将 f(n)定义为 g(n)和启发式 h(n)的和，其中 g(n)是到达节点 n 的路径代价，h(n)是从节点 n 到目标的路径代价。如果 h(n)没有高估真实代价，则 A*是完备的和最优的。

常用的启发式函数 h(n)包括错位棋子数和曼哈顿距离，本文实现了基于这两种启发式函数的 A*搜索算法用来解决 N 码问题，另外，本文也使用了线性冲突启发式（linear conflicts heuristic[3]）解决这一问题。实验结果表明，使用线性冲突启发式的 A*搜索在时间和空间的开销上要少于其他两种启发式。

2. 问题描述

给定一个 N 码问题的初始状态，寻找到达目标状态的路径。程序从文件 npuzzle_in.txt 中读入 N 码问题的数据，并将问题的解决方案输出到文件 npuzzle_out.txt 中。例如，图 2 所示为输入文件和输出文件的示意。输入文件中，

第 1 行数字 n，n 的范围为{2,3,4,5,6}，表示问题为(n^2 - 1)码问题；随后的 n×n 个数字表示初始状态的棋盘；再随后的 n×n 个数字表示目标状态的棋盘，其中 0 表示空格。输出文件中，给出 N puzzle 问题的每一步应该怎么走。

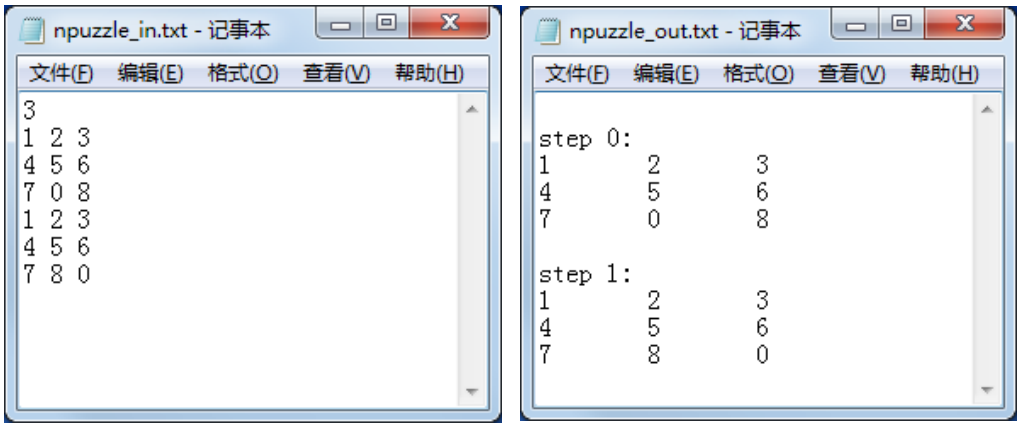

图 2. N 码问题的输入与输出示例

3. 问题解决

3.1 系统概况

本节主要介绍为了解决 N puzzle 问题，本系统是如何构成的概况。本系统包括测试样例生成，启发式函数，A*搜索几个部分。测试样例生成部分是用来随机生成文件 npuzzle_in.txt。启发式函数用来估计当前状态到目标状态的代价。A*搜索利用启发式函数搜索出从初始状态到目标状态的动作序列，可采纳的启发式函数可以保证 A*树搜索是搜索出的路径是最优的。

3.2 测试样例生成

本问题的测试样例 npuzzle_in.txt 是使用 Python 脚本随机生成的，例如生成一个 n=3 的 8 puzzle 问题，直接按如下方式调用 Python 脚本 TestCaseGeneration.py 即可。

```
python TestCaseGeneration.py 3
```

这个脚本的实现使用了 Python 的洗牌函数 shuffle，首先生成一个列表，包含 0 至(n^2 - 1)的所有整数。然后将其进行洗牌并以 n×n 的矩阵形式输出到 npuzzle_in.txt 作为初始状态。之后再洗牌输出到 npuzzle_in.txt 作为目标状态状

态。

3.3 启发式函数

本文中使用到的启发式函数包括错位棋子数，曼哈顿距离和线性冲突启发式。下面分别介绍。

错位棋子数是指当前状态的棋子不在正确的位置的数量。如图 3 所示的当前状态的错位棋子数启发式函数值为 8（因为所有的棋子都不在正确的位置）。

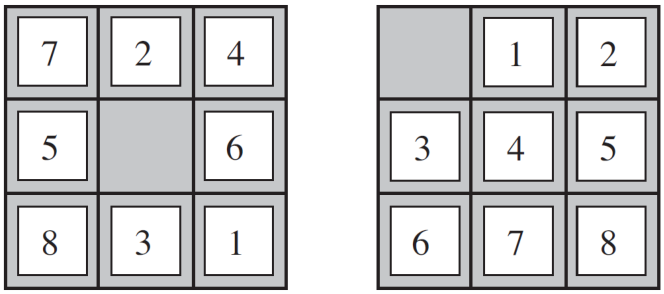

图 3. 8 码问题的当前状态与目标状态示例

曼哈顿距离启发式函数。对于每一枚棋子，其当前位置和目标位置之间会存在着一定的曼哈顿距离，所有棋子到达目标位置的曼哈顿距离之和即为曼哈顿距离启发式函数。例如图 3 中的 8 码问题，当前状态的棋子 1 至棋子 8 的曼哈顿距离之和为 3+1+2+2+2+3+3+2=18。

线性冲突启发式（linear conflicts heuristic[3]）函数。该启发式函数除了棋子到达目标位置的曼哈顿距离之外，还需要考虑线性冲突的问题。线性冲突定义为每行或者每列的某两个棋子的状态发生了互换，如图 4 所示为一个线性冲突启发式的示例。当出现线性冲突时，必须要有棋子移到另外一行上再移回该行，故需要在原有的曼哈顿距离启发式上再加 2。需要注意的是，有多少个线性冲突，就需要在曼哈顿距离启发式的基础上加上多少个 2。

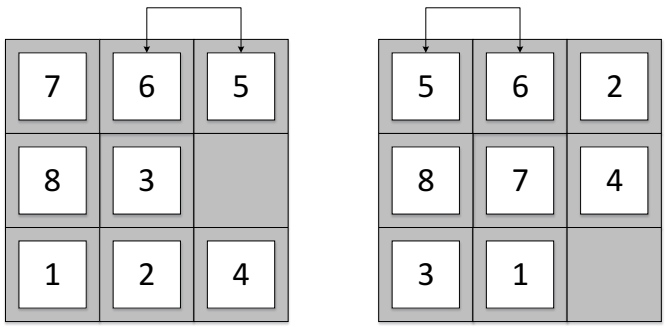

图 4. 线性冲突启发式示例

3.4 A*搜索

A*搜索的基本单元是一个命名为 state 的结构体，该结构体包含此时的棋盘状态，父状态，子状态，代价函数 g 的值，启发式函数 h 的值，所允许的动作。定义如下：

```
struct state{
    int config[10][10];       //Board configuration
    int g_cost;               //g_cost of the state
    int h_cost;               //h_cost of the state
    int allowed_operators[4]; //Left, Right, Up, Down
    struct state* next;
    struct state* parent;
};
```

A*搜索的实现如算法 1 所示。使用了 A*搜索的树搜索版本。注意搜索到目标状态时，需要调用一个过程（名为 create_solution）求出状态序列。该过程是一个递归调用的函数，通过递归求解到达其父状态的状态序列，直至初始结点。然后再回溯，依次打印出途径状态的棋盘。

算法 1. A*树搜索解决 N puzzle 问题

输入:

初始状态, init;

目标状态, goal;

输出:

打印出从初始状态到目标状态的状态序列;

```
1:   初始化: 状态链表states=NULL，临时状态temp=NULL
2:   将init放入states
3:   while True do
4:       if  states为空  do
5:           print “无解”
6:           return
7:       end if
8:       从states中删除g+h值最小的状态，并将该状态赋值给temp
9:       if  temp目标测试通过  do
10:          create_solution(temp)
11:          打印其他信息如耗时，消耗结点数
12:          return
```

```
13:         end if
14:         将temp所有子状态加入到states链表中
15:     end while
16:     return
```

```
过程: create_solution (state temp)
1:      if temp==init do
2:          print temp->config
3:          return
4:      else
5:          create_solution temp->parent
6:          print temp->config
7:          return
8:      end if
```

4. 实验

使用 PC 机 DELL Inspiron 3847，使用 C 语言在 Windows 7 平台上实现 A* 算法求解 N puzzle 问题。使用的 IDE 为 CodeBlocks 16.01。使用 Python 脚本语言生成测试用例，Python 版本 2.7.13。

4.1 程序正确性验证

使用图 3 中示例的 8 码问题对程序进行正确性验证，这里的启发式函数分别使用了使用曼哈顿距离线性冲突启发式。运行的结果如图 5 所示，同时显示出了展开的节点数和生成的节点数及运行时间。另外，程序生成了名为 npuzzle_out.txt 的解文件，这一文件如图 6 所示。

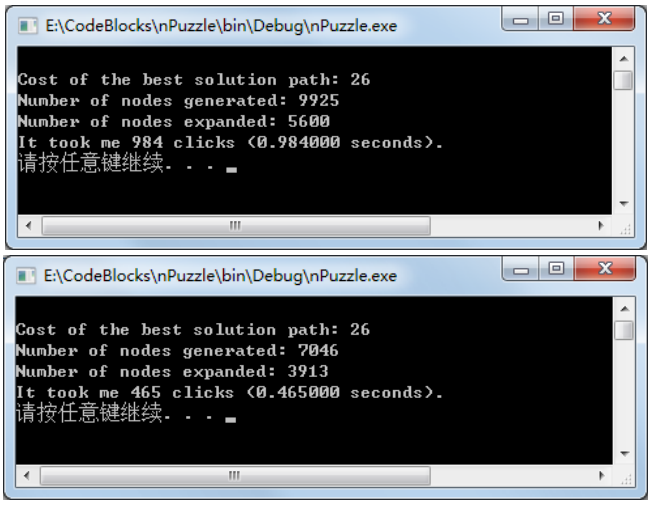

图 5. 分别使用曼哈顿距离和线性冲突启发式运行图 3 的 8 码问题

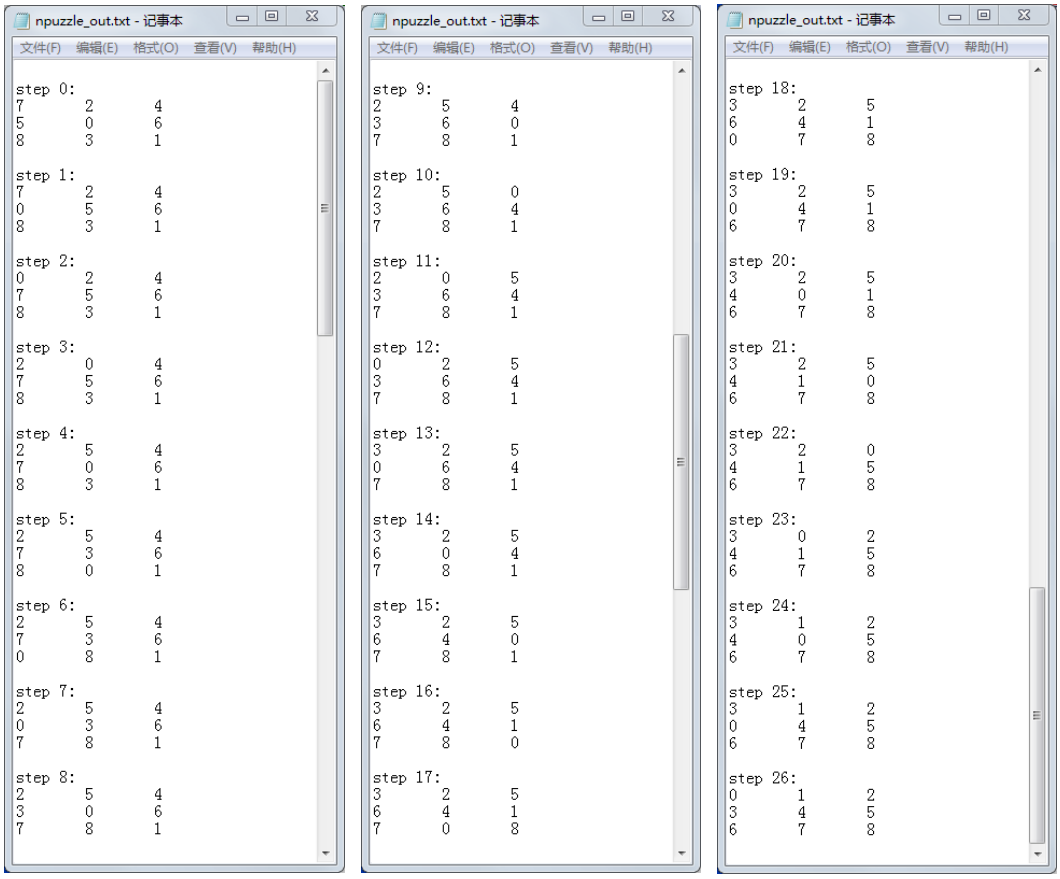

图 6. 程序正确运行生成的 npuzzle_out.txt

从图 6 可以看出，A*算法能生成正确的解，且使用不同的可采纳启发式都能生成同一个解。但是使用不同的启发式，在程序的运行效率和空间消耗上区别很大。图 5 可以明显看出，使用线性冲突启发式的运行过程需要消耗的时间和空间更少。另外，使用错位棋子数启发式的 A*算法不能在可接受时间内生成该问题的解。下一节需要对使用这三个启发式函数的 A*算法进行时间消耗和空间消耗上的评测以确定哪一个启发式函数解决 N 码问题更有效。

4.2 启发式函数的评测

本文中使用到的启发式函数包括错位棋子数，曼哈顿距离和线性冲突启发式。在满足可采纳的基础上，通常认为启发式函数值越大，在 A*搜索中效果越好。为了验证这一结论，对不同的$n \in \{2,3,4,5,6\}$进行了实验（实验的代码和测试用例见附件）。表 1 至表 5 分别给出了不同的 N 码问题的时间和空间代价。可以看出，使用线性冲突启发式的搜索算法在效率和空间利用率上更优于错位棋子数和曼哈顿距离启发式。

使用 A*搜索解决 N puzzle 问题有时可能会出现无法在可接受的时间内获得问题

的最优解的情况。这种情况一般出现在 n 较大且解的步数较多的情况（例如初始状态和目标状态都是随机生成的）。甚至会出现即使使用线性冲突启发式，仍然不能短时间内获得问题的解。

总结各类启发式的表现，可以发现：在相同的状态下，可采纳启发式的值越大，搜索的效率越高。

表 1. 使用不同的启发式解决 3 Puzzle 问题

启发式	运行时间/秒	生成节点数	展开节点数
错位棋子数	0	5	3
曼哈顿距离	0	5	3
线性冲突	**0**	**5**	**3**

表 2. 使用不同的启发式解决 8 Puzzle 问题

启发式	运行时间/秒	生成节点数	展开节点数
错位棋子数	0.12	3723	2133
曼哈顿距离	0.006	683	375
线性冲突	**0.004**	**602**	**333**

表 3. 使用不同的启发式解决 15 Puzzle 问题

启发式	运行时间/秒	生成节点数	展开节点数
错位棋子数	无法在可接受的时间内获得问题的最优解		
曼哈顿距离	0.079	2395	992
线性冲突	**0.048**	**1927**	**802**

表 4. 使用不同的启发式解决 24 Puzzle 问题

启发式	运行时间/秒	生成节点数	展开节点数
错位棋子数	无法在可接受的时间内获得问题的最优解		
曼哈顿距离	5.392	15268	5952
线性冲突	**0.264**	**3463**	**1241**

表 5. 使用不同的启发式解决 35 Puzzle 问题

启发式	运行时间/秒	生成节点数	展开节点数
错位棋子数	无法在可接受的时间内获得问题的最优解		
曼哈顿距离	无法在可接受的时间内获得问题的最优解		
线性冲突	**11.138**	**19427**	**7022**

5. 结论

本文通过使用 A*搜索算法解决 N 码问题。A*算法从有限的状态空间中搜索从初始状态到达目标状态的序列，A*算法的启发式能够指导算法朝哪个方向搜索。本文将三种启发式运用到 A*算法中，分别是错位棋子数，曼哈顿距离和线性冲突启发式。

实际实验过程中测量了各个启发式在搜索过程中的时间和空间开销，发现使用线性冲突启发式的 A*具有更高的效率和更少的时间开销。

但本文使用的方法也存在着一些问题，如求解太复杂的 N 码问题时仍然无法在可接受的时间内得到解。有一些改进可能会提升该算法，如使用双向搜索版本的 A*更快找到解，或使用迭代加深 A*减少空间开销。需要在以后的工作中进一步探索。

参考文献

[1] Klimaszewski J. The efficiency of the A* algorithm’s implementations in selected programming languages[J]. Journal of Theoretical and Applied Computer Science, 2014, 8(2): 63-71.

[2] Slocum J, Sonneveld D. The 15 Puzzle: How it Drove the World Crazy: the Puzzle that Started the Craze of 1880; how Amercia's Greatest Puzzle Designer, Sam Loyd, Fooled Everyone for 115 Years[M]. Slocum Puzzle Foundation, 2006.

[3] Korf R E, Taylor L A. Finding optimal solutions to the twenty-four puzzle[C] //Proceedings of the national conference on artificial intelligence. 1996: 1202-1207.