浙江大学

Artificial Intelligence
n-Puzzle Problem
Project Report

2020~2021 春夏学期 2021 年 3 月 29 日

Categories

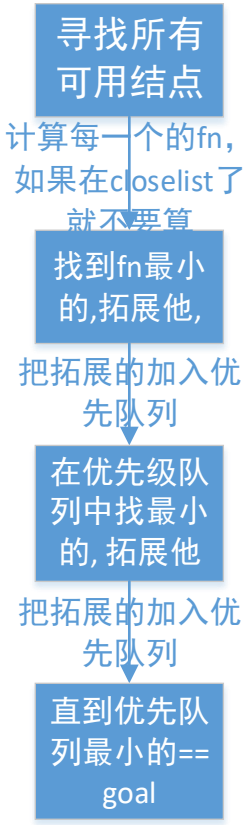
Chapter 1: Introduction

Background and Our goals:
This program asks implement an A-star search algorithm to solve 15-puzzle problem.

Chapter 2: Algorithm Specification

1.A good heuristic for A-Star with the 15 puzzles is the number of squares that are in the wrong location. Because you need at least 1 move per square that is out of place, the number of squares out of place is guaranteed to be less than or equal to the number of moves required to solve the puzzle, making it an appropriate heuristic for A-Star.
2. Sum of distances of the tiles from their actual positions (Manhattan distance) – sum of horizontal and vertical distances, for each tile.H(n) = Manhattan distance, |Xdst -Xsrc | +|Ydst-Ysrc |, it converges faster than the algorithm above.
3. G(n) = 1 because we move one squares at once
4. States corresponding to the same board position are enqueued on the priority queue（open list） many times. To prevent unnecessary exploration of useless states, when considering the neighbors of a state, don't enqueue the neighbor if its board position is the same as the previous（close list） state. We can use a Hashmap storing processed state to achieve this.
5. using Min Priority Queue data structure so that we can get the next state to examine quickly. It can be done without Min Priority Queue as well – but that would be too slow.

1. 获取第一个开始状态。
2. 获得所有可能的状态计算 fn。
3. 如果新状态 fn 比 在开放状态列表中的第一个好，就添加这新状态(开放列表是空的就加入)
4. 在已关闭列表中添加已处理的 state。
5. 选择在开放列表中成本最低的下一个状态(优先队列就是第一个)
6. 开始重复步骤从 1 到 6，除非你进入最终状态，或没有更多的状态来检查（在这种情况下，不存在解决方案）。
7. 一旦你有最终状态，回溯到开始节点，这将是找到解决方案的最佳路径。



Chapter 3: Testing Results

Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz　　1.80 GHz 运行时 cpu 占用率 25%		
阶数	100	200
1	1.56	19.73
2	1.78	17.8
3	1.31	over 60s
4	1.88	over 61s
5	over13秒	59.22
6	1.54	over120s
7	1.28	over120s
8	over33秒	over120s
9	1.33	over120s
10	1.4	over120s

Chapter 4: Analysis and Comments

未来期待的改进:
1. 我觉得 close list 用 set 在结点多的时候可以更快.但是我不太熟悉可能会很多错误所以先不实现.
2. IDA*,解决内存问题是基于迭代加深的 A*算法。并且不需要判重，不需要估价排序，用不到哈希表，空间需求量变得超级少。启发函数用曼哈顿距离。在找最优解的时候，对超过目前最优解的地方进行剪枝，可导致搜索深度急剧减少。再 check 一下不要回到上个状态。
3.一般的 15 码样例都能解决，但是对于 15 码最复杂的一些样例，仍然需要三四十分钟才能解出答案。还有什么可以优化吗？
A.Felner,R.Korf 的 Disjoint Pattern Database Heuristics，不相交模式数据库启发式算法。Additive Pattern Database Heuristics ——E.Korf，A.Felner，S.Hanan。这是一中新的设计更精确的可容许启发式评价函数的方法，该方法基于模式数据库。

Problems:

问题 1 怎么回溯 get path?

想到一个方法是用父节点一步步上去， 但是 state 保存在哪里呢？
就是 current state = current_state.pre_state， 如同一个链表

问题 2 怎么自定义优先队列比较？

解决方法: 使用重载方法__lt__，__lt__是 python 中用于进行特定比较的方法。
例如

```
def __lt__(self, other):
    if self.a == other.a:
        return self.b < other.b #1 小
    else:
        return self.a < other.a #小
```

问题 3 怎么保存父节点？
每个结点可以保存上一个结点，但是去哪里找所有的结点呢？

问题 4 怎么探索每个 state?

```
s1.state =
```

这样可能会出错。一开始 new 一个对象然后赋值可以不可以。

问题 5 怎么计算每个的 Manhattan 距离?
解决方法:

```
for i in range(1, child_state.square_size ** 2):
    dst_row, dst_col = dst_state.num_pos(i)
    chi_row, chi_col = child_state.num_pos(i)
    manhattan += abs(dst_row - chi_row) + abs(dst_col - chi_col)
```

问题 6 End of statement expected
解决方法: print(x) 代替 print x

问题 7: 为啥他在 close list 要 return？ 直接判断有了不就跳过吗？
原因: 是的,确实是直接跳过.

问题 8: pycharm 无法最大化, pycharm 最小化打不开。
解决方法: 重装 pycharm 也不行, 换个项目就可以了.

问题 9: ValueError: operands could not be broadcast together with shapes (0,) (3,)
维度不同, 不能相加, 因为 main 函数是 4 维, 而 puzzle 一开始是 3 维度
改了还是有错
ValueError: operands could not be broadcast together with shapes (2,4) (1,2)
原因: 这个加法是不是不能用于 numpy 的 array, numpy.zeros, 和 list 不一样
4:4 有问题, 3:3 没问题
解决方法: 调用它的 oncemove 函数

问题 10:死循环 in deepcopy y = copier(memo)
RecursionError: maximum recursion depth exceeded while calling a Python object
这是因为 stackoverflow 了, 回溯太多次了. 这可能是算法的问题, 也可能是代码里面的退出条件没加
后来它不报错了, 但是跑两三分钟都跑不出来,
一步步看看, start 复制好像是对的. Move list 没有用到, once move 也没有问题

```
if col < curr_state.square_size :
```

这里好像有问题, 其实是不行的, 这里会做很多不必要的操作, 进去判断了 move 产生了 state 之后发现是不行的,
其他三个方向好像是对的.

```
for i in range(child_state.square_size ** 2):
```

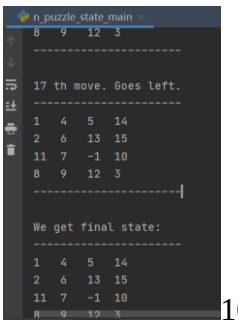
这里也不对, 应该从 1 开始, range 自动从 0 开始, 但是只是变慢一两步, 不会加上错误的值.
Mahattam 距离更新好像看起来也没错

```
然后就是in_list, match_state = state_in_list(child_state, open_list)
```

上面 child state 的 g 和 h 已经更新了, 所以应该根本不可能找到一样的.
我试了试, 10 步是可以的哈哈哈哈哈. 答案是走了 3 步.
然后 100 步就跑 2 分钟以上出不来. 有的时候它就出来的很快 .
平均耗时大概 30 秒以上吧.
在使用深度复制时，这总是需要注意的："递归对象（直接或间接包含引用自己的复合对象）可能会导致递归循环
应该是因为他的 pre state 一直回溯, 形成了一个循环.
https://stackoverflow.com/questions/42570634/recursion-error-with-python-3-copy-deepcopy

但是我不知道怎么解决他.
解决方法:
因为 open list append 我前面改成了 heapq.heappush(open_list, child_state), 后面又没有 push
heapq.heappush(

问题 11: AttributeError: 'NoneType' object has no attribute 'state' 50 步,11 秒,4 秒 16 步,困难的两个跑了一分钟也没出来,简单的 2 秒 10 步,2 秒.



100 步简单可以 4 秒左右,慢的跑了五分钟也没出来.

原因: 因为 open list append 我前面改成了 heapq.heappush(open_list, child_state), 后面又没有 push heapq.heappush(

问题 12: TypeError: object of type 'NoneType' has no len() 因为没有 moves, 没有找到动作.
为啥 openlist 空了还是没找到呢.
原因: child state 应该加的没有加. 把这个判断注释掉.

```
# if open_list.__len__() > 0:
#         fno = open_list[0].g + open_list[0].h
#         fnc = child_state.g + child_state.h
#         if fnc < fno:
#                 heapq.heappush(open_list, child_state)
# else:
heapq.heappush(open_list, child_state)
        # open_list.append(child_state)
```