bonus.md

Bonus points assignment: the complexity of the 8-puzzle

First part

Is this the total number of possible configurations for the 8-puzzle? If not, what is the total number of possible configurations?

181440 is not the total number of possible configurations for the 8-puzzle, the correct number is 362880, which is two times as many.

Is it possible to solve the 8-puzzle starting from any initial configuration? If not, prove what is the number of states for which the goal state is unreachable and demonstrate a simple example of a starting position for which the puzzle is unsolvable. Perform the same analysis for the 15-puzzle

No, it is not solvable for all states, cause the state space is divided into two disjoint sets, both with 181440 states. A starting position that will not succeed is "" if the goal is the "". The demonstration program map (modified puzzle state space definition) is saved under "./maps/3x3_puzzle_unsolvable.txt", but it has no chance of winning because there are no transitions from that states written in the state space definition file, so it cannot move anywhere even at the first step.

To prove the idea of two disjoint sets of states in the 8 puzzle state space, lets take a more mathematical approach to the problem.

Proof for the 8-puzzle

First, let's turn the puzzle state into one row of numbers by taking the numbers as they show up in the state from right to left, from top to bottom. Then the state 123_45#_678 would turn into 12345678. Let's call this notation R. Let N denote the number of inversions that occur in R, where inversion is if a higher number is before a lower one. For example, this R has 3 inversions 32145678, namely 3->1, 3->2 and 2->1.

With these things defined, let's look at how N can change if we move tiles. By moving any tile right or left, N does not change because R does not change. Only by moving tiles up or down can N change. By examining the moves carefully, one will notice that by moving the tile up or down, the number that has been moved would jump over exactly two other numbers, So if we move 5 up in this R -- 12345678, then the new R would be 1234**67**589. By jumping over two numbers the shift in N will either be 2 (if the two numbers were both higher or lower than 5) or 0 (if one number was higher and the other lower).

Because up, down, left and right are all permitted moves, one can surely conclude that N would be of same parity the whole time - either odd or even. If the start state was even, then N would always be even. If the start state was odd, N would always be odd. If the goal state was of different parity, it cannot be reached.

Therefore, the state space will be divided into two parts, one of even parity and the other of odd. This makes the state space disjoint in the sense that it is made up of two unconnected graphs of states. As same moves are permitted in both disjoint sets, one can reach the same number of states in both of them. In other words, each set must have the same number of states. As the total number of states is 9!=362880, each of the disjoint sets will have 9!/2=181440 states.

Proof for the 15-puzzle

Analysing the 15 puzzle, one can come to an similar conclusion - the set is again divided into two disjoint sets. A brief proof follows, for more information and the a mathematically formal proof, head to:

- https://www.cs.bham.ac.uk/~mdr/teaching/modules04/java2/TilesSolvability.html
- http://kevingong.com/Math/SixteenPuzzle.html

For a 4x4 grid, the following statement must hold: *polarity of inversion is the always either the same as the polarity of the row where the empty tile is or always different.* For all legal moves this statement will hold. Moving the tiles left or right changes neither the polarity of inversions or the row of the empty tile. Moving the tiles up and down changes the row number of the empty tile by ±1, whereas the number of inversions N is changed by either ±1 or ±3. The later is because the tile that has been moved will jump over three other tiles in the R notation, and each of the tiles could have been higher or greater than the tile moved, which gives (±1±1±1), and all the combinations that can be made from this equation are odd (±1 and ±3). Therefore the polarity of both N and the empty tile row number change, and the statement from the start holds true.

In the initial configuration of the puzzle (aka in the start state), N will either be even or odd, and the empty tile will either ne on an even or odd row number. The (even, even) and (odd, odd) can lead one into another as well as (even, odd) and (odd, even). Therefore there are at least two disjoint sets of states and the 15 puzzle wont be always solvable. For a proof that there are only these two disjoint sets and not more of them (like 4 or 6), head to the links above.

Second part

(1)

a. Optimism check

As I use dijkstra to determine the oracle, the complexity (using heapq as min priority queue) is: `O(|E| + |V|log|V|) --> O(|S|*b + |S|*log|S|)`

b. Consistency check

I've written two variants

1. variant: The written algorithm visits all states that can be reached from the start_state and from the goal_states, each time adding the newly seen states to a set and looking at the consistency relationship between the successor and predecessor. Therefore, all states are visited once and (directed) edges between them are also processed once. The time complexity of this solution is `O(|S| + |E|)`
2. variant: All states are given to the check function, so only visiting all of them and checking the relationships is needed. It is of same time complexity as 1. variant, but a bit faster.

(2)

The original implementation I had written was between 2 and 3 seconds for all heuristic check algorithms. After tweaking and optimising the code and writing a demo benchmark `demo_3x3_my_heuristic_benchmark.py`, below are the resulting statistics of execution times after the performance tweaks (data saved in benchmark-results.txt):

Optimism check:

- Number of samples (N): 100
- Mean (Average) (μ): 1.2968436861
- Sample Standard Deviation (s): 0.095950399329

Consistency check (variant 1):

- Number of samples (N): 100
- Mean (Average) (μ): 0.77355415821
- Sample Standard Deviation (s): 0.019559864621

Consistency check (variant 2):

- Number of samples (N): 100
- Mean (Average) (μ): 0.56870261669
- Sample Standard Deviation (s): 0.012727823504

Third part

Can you think of a simple example showing that this heuristic isn’t optimistic?

A simple example to show that this heuristic is not optimistic is to look at a first move made in the game, like this one **123_456_78# --> 123_456_7#8**. The real (oracle) cost of solving the puzzle if it is left in this state is 1, because only one move is needed to move the tile 8 back to its place. But by counting the moves that the empty tile made, in some cases one counts twice as many moves than is really needed (78 of these times, the heuristic really fails to be optimistic).

Which simple change should be done to the heuristic to make it both optimistic and consistent?

Dividing it by 2 would solve many problems. The demonstration file called `"demo_3x3_heuristic_quick_fix.py"` does exactly that. Here is the output for that:

```
Loaded 181440 states with 483840 transitions
Start state is: 876_543_21x
Goal states are (always the same): {'123_456_78x'}

----> A*
States visited --> 160904
Number of steps -->  31
Found path of cost -->  30.0
876_543_21x-->
876_543_2x1-->
876_5x3_241-->
8x6_573_241-->
x86_573_241-->
586_x73_241-->
586_273_x41-->
586_273_4x1-->
586_2x3_471-->
5x6_283_471-->
56x_283_471-->
563_28x_471-->
563_2x8_471-->
5x3_268_471-->
x53_268_471-->
253_x68_471-->
253_468_x71-->
253_468_7x1-->
253_468_71x-->
253_46x_718-->
253_4x6_718-->
253_416_7x8-->
253_416_x78-->
253_x16_478-->
253_1x6_478-->
2x3_156_478-->
x23_156_478-->
123_x56_478-->
123_456_x78-->
123_456_7x8-->
123_456_78x

----> Was the used heuristic optimistic?
Delta t: 1.3960247039794922 seconds
Heuristic is optimistic for all states that could be visited from the goal state.
----> Was the used heuristic consistent?
Delta t: 0.8942818641662598 seconds
Heuristic is consistent
----> Was the used heuristic consistent? (Faster check with all states given)
Delta t: 0.6918761730194092 seconds
Heuristic is consistent
```

Try to come up with at least two new good heuristic functions. Measure the number of visited states for the A* algorithm for each heuristic on at least 10 different initial configurations, including the default one and store the number of visited states for each configuration.

I have implemented these two heuristic functions:

1. L0 distance from goal state
2. L1 distance from goal state

Here is the sample output for both of them when run:

```
####################
### Heuristic L0 ###
####################
----> A*
States visited --> 92912
Number of steps -->  31
Found path of cost -->  30
876_543_21x-->
876_54x_213-->
876_5x4_213-->
```

```
876_x54_213-->
876_254_x13-->
876_254_1x3-->
876_2x4_153-->
876_24x_153-->
876_243_15x-->
876_243_1x5-->
876_2x3_145-->
8x6_273_145-->
x86_273_145-->
286_x73_145-->
286_173_x45-->
286_173_4x5-->
286_1x3_475-->
2x6_183_475-->
26x_183_475-->
263_18x_475-->
263_1x8_475-->
2x3_168_475-->
x23_168_475-->
123_x68_475-->
123_468_x75-->
123_468_7x5-->
123_468_75x-->
123_46x_758-->
123_4x6_758-->
123_456_7x8-->
123_456_78x
----> Was the used heuristic optimistic?
Delta t: 3.9540066719055176 seconds
Heuristic is optimistic for all states that could be visited from the goal state.
----> Was the used heuristic consistent?
Delta t: 7.13589334487915 seconds
Heuristic is consistent
```

```
####################
### Heuristic L1 ###
####################
----> A*
States visited --> 7353
Number of steps -->  31
Found path of cost -->  30
876_543_21x-->
876_543_2x1-->
876_5x3_241-->
876_53x_241-->
876_531_24x-->
876_531_2x4-->
876_531_x24-->
876_x31_524-->
x76_831_524-->
7x6_831_524-->
736_8x1_524-->
736_81x_524-->
73x_816_524-->
7x3_816_524-->
713_8x6_524-->
713_826_5x4-->
713_826_x54-->
713_x26_854-->
x13_726_854-->
1x3_726_854-->
123_7x6_854-->
123_756_8x4-->
123_756_84x-->
123_75x_846-->
123_7x5_846-->
123_745_8x6-->
123_745_x86-->
123_x45_786-->
123_4x5_786-->
123_45x_786-->
123_456_78x
```

```
----> Was the used heuristic optimistic?
Delta t: 4.482226610183716 seconds
Heuristic is optimistic for all states that could be visited from the goal state.
----> Was the used heuristic consistent?
Delta t: 8.56673789024353 seconds
Heuristic is consistent
```

Here are the results of running both of them on ten different configurations that I have picked by hand (demonstration program is `"demo_3x3_my_heuristic_benchmark.py"`):

Heuristic L0:

States visited	Cost
2373	20
1232	18
9143	23
2549	20
4999	22
162	14
76655	29
14130	24
1547	19
41112	27

Heuristic L1:

States visited	Cost
457	20
203	18
657	23
196	20
549	22
30	14
5049	29
263	24
221	19
1989	27