Writeup for EECS 391 Programming Assignment 1

Jiaqi Yang (jxy530)

09/27/2019

---

1. Code Design

State

State is an abstract class that represents a state for a puzzle. In the 8-puzzle game, a state is represented by class EPState, which inherits State abstract class. The core of EPState class is stateList, an ArrayList of Integer that represents 8-puzzle state, which Integer 0 is used to represent blank tile. EPstate provides eight methods as shown below.

```
    //Set the stateList to the goal state of the puzzle
    public abstract void setGoalState();

    //Test if the current state is the goal state
    public abstract boolean isGoalState();

    //Set the stateList to represent the state of input
    public abstract void setStateList(String state);

    //Validate the input puzzle representation
    public abstract boolean isValidState(String state);

    //Print the current puzzle state
    public abstract void printState();

    //Get the hash code of the current state in order to distinguish different
states
    public abstract int getHash();

    //Get a new instance of State
    public abstract State copyState();

    //Validate if the current state is solvable
    public abstract boolean isSolvable();
```

Furthermore, State abstract class provides a function to randomly move from the goal state by passing number of steps. This is the implementation of randomizeState command. The seed used for Random class is 1.

Move

Move is an interface that represents a move in a given state. In the 8-puzzle game, all possible moves are up, down, left, right; the corresponding classes that implements Move interface are EPMoveUp, EPMoveDown, EPMoveLeft, EPMoveRight. All these classes provide three methods as shown below.

```
    //Allow the tile to be moved in different directions
    State move(State state);

    //Get the identifier of the move direction, which is up, down, left, right
    String getMoveName();

    //Validate if the movement is legal
    boolean isLegalMovement(State state);
```

Heuristic

Heuristic is an interface that provides the functionality of calculating the heuristic value of a given state. For 8-puzzle, there are 2 different heuristic function required: h1, which is the number of misplaced tiles, and h2, which is the sum of the distance of the tiles from their goal positions; the corresponding classes that implements Heuristic interface are EPMisplacedHeuristic and EPDistanceHeuristic. All these classes provide three methods as shown below.

```
    //Calculate the heuristic value of a given state
    double calculate(State state);

    //Return the String identifier of current heuristic function
    String getName();
```

Puzzle

Puzzle abstract class is the core implementation of Program Assignment 1. It provides the functionality to execute commands, read input File, and include the implementation of A-star algorithm and adapted local beam search algorithm. The major functional methods are described below.

- ExecuteCommand method takes input String commands and call the corresponding methods to execute them.
- solveAstar method takes input Heuristic, specified by the command, and boolean shouldPrint. A-star algorithm is implemented with PriorityQueue. Until the algorithm finds the goal state, it keeps polling the next unexplored State and put its successor State into the PriorityQueue. The core implementation is attached.

```
ArrayList<Integer> exploredNodesHash = new ArrayList<>();
boolean reachGoalState = false;
SearchNode goalStateSearchNode = null;
PriorityQueue<SearchNode> priorityQueue = new PriorityQueue<>();

priorityQueue.add(new SearchNode(state, heuristic.calculate(state)));

while(reachGoalState == false && exploredNodesHash.size() < maxNodes){

    SearchNode searchNode = priorityQueue.poll();
    while(exploredNodesHash.contains(searchNode.getState().getHash())){
        searchNode = priorityQueue.poll();
            if(searchNode == null)
                return new Solution(exploredNodesHash.size(), -1);
    }

    exploredNodesHash.add(searchNode.getState().getHash());
```

```
    if(searchNode.getState().isGoalState()){
        goalStateSearchNode = searchNode;
        reachGoalState = true;
    }

    else{
        for(Move move: getMoveList()){
            if(move.isLegalMovement(searchNode.getState())){
                State movedState = move.move(searchNode.getState());

                if(!exploredNodesHash.contains(movedState.getHash())){

                    ArrayList<Move> movesToMovedStateNode =
                            searchNode.getMovesToCurrentNode();
                    movesToMovedStateNode.add(move);
                    priorityQueue.add(new SearchNode(movedState,
heuristic.calculate(movedState),
movesToMovedStateNode,
       searchNode.getCostToCurrentNode()+1));
                }
            }
        }
    }
}
```

- solveBeam method takes int numberOfState, which is the number of beams, and boolean shouldPrint. The adapted local beam algorithm is designed that
    - First, Initialize all local variable. The heuristic function used in this algorithm is h2, sum of the distance of the tiles from their goal positions.
    - Second, put the starting state and numberOfState - 1 states that randomly move 10 steps from the starting state. These states fill the bestkNodes, which is the ArrayList that stores the k nodes with smallest heuristic value.
    - Third, until we find the goal state or reach the upper limitation of node exploration, we put all the successors of those states in the bestkNodes into a temporary PriorityQueue. After finding all successors, we clear bestkNodes list and preferentially put unexplored successors with highest h2 value into bestkNodes.

EPPuzzle class extends Puzzle abstract class and includes specific variables and methods for 8-puzzle game. It is the class where the main method for 8-puzzle game is located. It takes the file name in the same folder, read and execute the commands line by line. It contains two HashMap that stores the Heuristic and Move for 8-puzzle and multiple methods to return the corresponding Heuristic and Move based on the input String name.

SearchNode

SearchNode class presents a A-star or local beam search tree node, including the information of g(n), h(n), moves from the start state, and the current state. The cost from the starting state to current state is not used in local beam search, and thus set to 0 for those nodes. SearchNode class also implements Comparable interface in order to support the soring in PriorityQueue.

Solution

Solution class presents the information about a solution of the puzzle, including number of explored nodes and steps getting to goal state. This class is designed for the experiment part of the programming assignment in order to store the results of each run.

---

2. Code Correctness

The only main method in the program is in the EPState class. To run the program, please run the main method and pass the file name as the only argument.

The following instructions are in the file named SimpleTestFile.txt within my submitted zip file.

```
Commands:
setState b12 345 678
printState

Output:
The input state is b12 345 678
The current 8-puzzle state is
    1   2
3   4   5
6   7   8
```

These two lines tests the setState command and printState command. The output meets the expectation, which is exactly same as the input String representation.

```
Commands:
maxNodes 1000
move right
printState
randomizeState 2
printState

Output:
The maximum number of nodes is set to 1000
The input movement is right
The current 8-puzzle state is
1       2
3   4   5
6   7   8
The number of random moves from the goal state is 2
The current 8-puzzle state is
3   1   2
6   4   5
    7   8
```

These two lines test the maxNodes command, move command, and randomizeState command. The output meets the expectation: the blank tile moves right from the goal state, and then from the goal state, the blank tile moves down and then down again.

```
Command:
solve A-star h1

Output:
The State starting with: The current 8-puzzle state is
3   1   2
6   4   5
    7   8
The next step is to move up. The current 8-puzzle state is
3   1   2
    4   5
6   7   8
```

```
The next step is to move up. The current 8-puzzle state is
    1   2
3   4   5
6   7   8
We reach goal state with 2 steps
[up, up]
```

Then we test the solve A-star h1 command. The output is optimal: the shortest path to the goal state is up and up. This fits the expected behavior of A-star algorithm.

```
Commands:
randomizeState 10
printState
solve A-star h2

Output:
The number of random moves from the goal state is 10
The current 8-puzzle state is
1   2
3   4   5
6   7   8
Solve the puzzle with A-star and heuristic function h2
The State starting with: The current 8-puzzle state is
1   2
3   4   5
6   7   8
The next step is to move left. The current 8-puzzle state is
1       2
3   4   5
6   7   8
The next step is to move left. The current 8-puzzle state is
    1   2
3   4   5
6   7   8
We reach goal state with 2 steps
[left, left]
```

Then we test the solve A-star h2 command. The output is optimal: the shortest path to the goal state is left and left. This fits the expected behavior of A-star algorithm.

```
Commands:
randomizeState 100
printState
solve beam 30

Output:
The number of random moves from the goal state is 100
The current 8-puzzle state is
5   2   8
6       1
4   3   7
Solve the puzzle with local beam search with state number 30
The State starting with: The current 8-puzzle state is
5   2   8
6       1
4   3   7
The next step is to move down. The current 8-puzzle state is
```

```
5   2   8
6   3   1
4       7
The next step is to move up. The current 8-puzzle state is
5   2   8
6       1
4   3   7
The next step is to move left. The current 8-puzzle state is
5   2   8
    6   1
4   3   7
The next step is to move down. The current 8-puzzle state is
5   2   8
4   6   1
    3   7
The next step is to move right. The current 8-puzzle state is
5   2   8
4   6   1
3       7
The next step is to move left. The current 8-puzzle state is
5   2   8
4   6   1
    3   7
The next step is to move right. The current 8-puzzle state is
5   2   8
4   6   1
3       7
The next step is to move up. The current 8-puzzle state is
5   2   8
4       1
3   6   7
The next step is to move right. The current 8-puzzle state is
5   2   8
4   1
3   6   7
The next step is to move left. The current 8-puzzle state is
5   2   8
4       1
3   6   7
The next step is to move right. The current 8-puzzle state is
5   2   8
4   1
3   6   7
The next step is to move up. The current 8-puzzle state is
5   2
4   1   8
3   6   7
The next step is to move left. The current 8-puzzle state is
5       2
4   1   8
3   6   7
The next step is to move down. The current 8-puzzle state is
5   1   2
4       8
3   6   7
The next step is to move left. The current 8-puzzle state is
5   1   2
    4   8
```

```
3   6   7
The next step is to move up. The current 8-puzzle state is
    1   2
5   4   8
3   6   7
The next step is to move right. The current 8-puzzle state is
1       2
5   4   8
3   6   7
The next step is to move down. The current 8-puzzle state is
1   4   2
5       8
3   6   7
The next step is to move left. The current 8-puzzle state is
1   4   2
    5   8
3   6   7
The next step is to move down. The current 8-puzzle state is
1   4   2
3   5   8
    6   7
The next step is to move right. The current 8-puzzle state is
1   4   2
3   5   8
6       7
The next step is to move right. The current 8-puzzle state is
1   4   2
3   5   8
6   7
The next step is to move up. The current 8-puzzle state is
1   4   2
3   5
6   7   8
The next step is to move left. The current 8-puzzle state is
1   4   2
3       5
6   7   8
The next step is to move up. The current 8-puzzle state is
1       2
3   4   5
6   7   8
The next step is to move left. The current 8-puzzle state is
    1   2
3   4   5
6   7   8
We reach goal state after searching 258 nodes.
We can get to the goal state by moving blank tile 26 times.
[down, up, left, down, right, left, right, up, right, left, right, up,
left, down, left, up, right, down, left, down, right, right, up, left, up,
left]
```

Lastly we test solve beam command. Obviously, the local beam search is not optimal, but it will provide a path to the goal state. In this case, we take 26 steps to get to the goal state, which only takes 20 steps for A-star algorithm, and thus it is not optimal. The behavior fits the expected behavior of local beam search, and therefore it is correct.

---

3. Experiments

Control Experiments of the Code

Before diving into the experiments and discussion sections, I would like to present my output of control experiments. I implemented a method called getExperimentAndDiscussionInfo in EPState class, and the outputs will be self-explanatory. I will elaborate these outputs as answers for experiments and discussion sections.

```
Command:
getExperimentAndDiscussionInfo

Output:
Here we present all the information required for experiments and discussion
section.
--------------------------------------------------------------------------
----------
MaxNodes: 50
h1: Solved: 728, Solvable fraction: 0.40%, Time consumed: 7556, Average
path length: 8, Average node explored: 26
h2: Solved: 4453, Solvable fraction: 2.45%, Time consumed: 8168, Average
path length: 13, Average node explored: 31
local beam search (k = 2): Solved: 13432, Solvable fraction: 7.40%, Time
consumed: 9705, Average path length: 23, Average node explored: 37
local beam search (k = 3): Solved: 6656, Solvable fraction: 3.67%, Time
consumed: 9640, Average path length: 20, Average node explored: 37
local beam search (k = 5): Solved: 2582, Solvable fraction: 1.42%, Time
consumed: 10642, Average path length: 17, Average node explored: 38
local beam search (k = 10): Solved: 694, Solvable fraction: 0.38%, Time
consumed: 13190, Average path length: 15, Average node explored: 35
local beam search (k = 20): Solved: 320, Solvable fraction: 0.18%, Time
consumed: 18478, Average path length: 13, Average node explored: 31
local beam search (k = 30): Solved: 222, Solvable fraction: 0.12%, Time
consumed: 24363, Average path length: 12, Average node explored: 27
local beam search (k = 50): Solved: 194, Solvable fraction: 0.11%, Time
consumed: 32433, Average path length: 11, Average node explored: 26
local beam search (k = 100): Solved: 183, Solvable fraction: 0.10%, Time
consumed: 60845, Average path length: 11, Average node explored: 29
--------------------------------------------------------------------------
----------
MaxNodes: 100
h1: Solved: 1656, Solvable fraction: 0.91%, Time consumed: 16125, Average
path length: 10, Average node explored: 54
h2: Solved: 11483, Solvable fraction: 6.33%, Time consumed: 15414, Average
path length: 14, Average node explored: 57
local beam search (k = 2): Solved: 55507, Solvable fraction: 30.59%, Time
consumed: 17874, Average path length: 38, Average node explored: 66
local beam search (k = 3): Solved: 37432, Solvable fraction: 20.63%, Time
consumed: 19037, Average path length: 31, Average node explored: 71
local beam search (k = 5): Solved: 22007, Solvable fraction: 12.13%, Time
consumed: 20655, Average path length: 25, Average node explored: 74
local beam search (k = 10): Solved: 6276, Solvable fraction: 3.46%, Time
consumed: 24215, Average path length: 20, Average node explored: 76
local beam search (k = 20): Solved: 1579, Solvable fraction: 0.87%, Time
consumed: 30906, Average path length: 16, Average node explored: 71
local beam search (k = 30): Solved: 804, Solvable fraction: 0.44%, Time
consumed: 37237, Average path length: 14, Average node explored: 63
```

```
local beam search (k = 50): Solved: 560, Solvable fraction: 0.31%, Time
consumed: 50729, Average path length: 13, Average node explored: 59
local beam search (k = 100): Solved: 434, Solvable fraction: 0.24%, Time
consumed: 85886, Average path length: 12, Average node explored: 56
--------------------------------------------------------------------------
----------
MaxNodes: 500
h1: Solved: 9679, Solvable fraction: 5.33%, Time consumed: 144279, Average
path length: 13, Average node explored: 276
h2: Solved: 70675, Solvable fraction: 38.95%, Time consumed: 114685,
Average path length: 18, Average node explored: 256
local beam search (k = 2): Solved: 181409, Solvable fraction: 99.98%, Time
consumed: 41008, Average path length: 89, Average node explored: 167
local beam search (k = 3): Solved: 179233, Solvable fraction: 98.78%, Time
consumed: 47782, Average path length: 70, Average node explored: 188
local beam search (k = 5): Solved: 170811, Solvable fraction: 94.14%, Time
consumed: 60905, Average path length: 53, Average node explored: 215
local beam search (k = 10): Solved: 166086, Solvable fraction: 91.54%, Time
consumed: 80130, Average path length: 39, Average node explored: 264
local beam search (k = 20): Solved: 135552, Solvable fraction: 74.71%, Time
consumed: 123659, Average path length: 30, Average node explored: 328
local beam search (k = 30): Solved: 98186, Solvable fraction: 54.11%, Time
consumed: 160136, Average path length: 26, Average node explored: 360
local beam search (k = 50): Solved: 45005, Solvable fraction: 24.80%, Time
consumed: 212110, Average path length: 23, Average node explored: 382
local beam search (k = 100): Solved: 10225, Solvable fraction: 5.64%, Time
consumed: 293340, Average path length: 18, Average node explored: 365
--------------------------------------------------------------------------
----------
MaxNodes: 1000
h1: Solved: 19020, Solvable fraction: 10.48%, Time consumed: 480724,
Average path length: 15, Average node explored: 539
h2: Solved: 113043, Solvable fraction: 62.30%, Time consumed: 292001,
Average path length: 20, Average node explored: 442
local beam search (k = 2): Solved: 181440, Solvable fraction: 100.00%, Time
consumed: 41277, Average path length: 89, Average node explored: 167
local beam search (k = 3): Solved: 181440, Solvable fraction: 100.00%, Time
consumed: 47926, Average path length: 72, Average node explored: 192
local beam search (k = 5): Solved: 181409, Solvable fraction: 99.98%, Time
consumed: 64176, Average path length: 58, Average node explored: 237
local beam search (k = 10): Solved: 181359, Solvable fraction: 99.96%, Time
consumed: 86139, Average path length: 42, Average node explored: 293
local beam search (k = 20): Solved: 181112, Solvable fraction: 99.82%, Time
consumed: 141551, Average path length: 34, Average node explored: 401
local beam search (k = 30): Solved: 179536, Solvable fraction: 98.95%, Time
consumed: 207191, Average path length: 31, Average node explored: 493
local beam search (k = 50): Solved: 163654, Solvable fraction: 90.20%, Time
consumed: 348913, Average path length: 28, Average node explored: 623
local beam search (k = 100): Solved: 86819, Solvable fraction: 47.85%, Time
consumed: 634820, Average path length: 24, Average node explored: 742
--------------------------------------------------------------------------
----------
MaxNodes: 5000
h1: Solved: 66785, Solvable fraction: 36.81%, Time consumed: 10569355,
Average path length: 18, Average node explored: 2200
h2: Solved: 177702, Solvable fraction: 97.94%, Time consumed: 1431587,
Average path length: 21, Average node explored: 1051
```

```
local beam search (k = 2): Solved: 181440, Solvable fraction: 100.00%, Time
consumed: 41092, Average path length: 89, Average node explored: 167
local beam search (k = 3): Solved: 181440, Solvable fraction: 100.00%, Time
consumed: 47890, Average path length: 72, Average node explored: 192
local beam search (k = 5): Solved: 181440, Solvable fraction: 100.00%, Time
consumed: 63994, Average path length: 58, Average node explored: 237
local beam search (k = 10): Solved: 181440, Solvable fraction: 100.00%,
Time consumed: 85540, Average path length: 42, Average node explored: 294
local beam search (k = 20): Solved: 181440, Solvable fraction: 100.00%,
Time consumed: 140737, Average path length: 34, Average node explored: 401
local beam search (k = 30): Solved: 181440, Solvable fraction: 100.00%,
Time consumed: 206896, Average path length: 31, Average node explored: 500
local beam search (k = 50): Solved: 181440, Solvable fraction: 100.00%,
Time consumed: 360174, Average path length: 29, Average node explored: 673
local beam search (k = 100): Solved: 181440, Solvable fraction: 100.00%,
Time consumed: 839851, Average path length: 27, Average node explored: 1034
--------------------------------------------------------------------------
----------
MaxNodes: 10000
h1: Solved: 97333, Solvable fraction: 53.64%, Time consumed: 31708667,
Average path length: 19, Average node explored: 6478
h2: Solved: 181440, Solvable fraction: 100.00%, Time consumed: 2290539,
Average path length: 22, Average node explored: 1945
local beam search (k = 2): Solved: 181440, Solvable fraction: 100.00%, Time
consumed: 41092, Average path length: 89, Average node explored: 167
local beam search (k = 3): Solved: 181440, Solvable fraction: 100.00%, Time
consumed: 47890, Average path length: 72, Average node explored: 192
local beam search (k = 5): Solved: 181440, Solvable fraction: 100.00%, Time
consumed: 63994, Average path length: 58, Average node explored: 237
local beam search (k = 10): Solved: 181440, Solvable fraction: 100.00%,
Time consumed: 85540, Average path length: 42, Average node explored: 294
local beam search (k = 20): Solved: 181440, Solvable fraction: 100.00%,
Time consumed: 140737, Average path length: 34, Average node explored: 401
local beam search (k = 30): Solved: 181440, Solvable fraction: 100.00%,
Time consumed: 206896, Average path length: 31, Average node explored: 500
local beam search (k = 50): Solved: 181440, Solvable fraction: 100.00%,
Time consumed: 360174, Average path length: 29, Average node explored: 673
local beam search (k = 100): Solved: 181440, Solvable fraction: 100.00%,
Time consumed: 839851, Average path length: 27, Average node explored: 1034
```

> (a) How does fraction of solvable puzzles from random initial states vary with the maxNodes limit?

I implemented a method getAllSolvableStates in the class EPState, which returns all solvable state for 8-puzzle game. In the getExperimentAndDiscussionInfo method, I originally implemented the experiments from 8 different MaxNodes combinations: 100, 500, 1000, 5000, 10000, 50000, 100000, 500000 threshold; however, the running time for h1 becomes too long that it is impossible to finish all the states with h1 with my computer. So I ended up only collecting the information from the range 100 to 10000. The general knowledge from the data collected is summarized below.

- The performance of A-star algorithms fits the expectation: since they are hunting for the optimal path instead of some random path, they will solve less state than local beam.
    - At condition of maxNodes upper limit 50 -100, h1 cannot solve even 1% of solvable 8-puzzles, and h2 only solve 6.33% at best. This is poorly-performed compared to local

beam k =2, which solves 30% at best.
    - At condition of maxNodes upper limit 10000, h2 eventually solve 100% of solvable 8-puzzles, and h1 just hits 50%. In comparison, local beam k = 2 solves 100% within 500 maxNodes, and even local beam k =100 solves 100% within 5000 maxNodes.
- Since the limitation in this case is maxNodes limit, local beam search with smallest k number, in this case k = 2, is the most efficient. And generally, local beam search is much more capable of solving 8-puzzle given the fact that it is not looking for an optimal path.
    - At condition of maxNodes upper limit 500, the local beam search with k =2 solves 99.98% of solvable 8-puzzles, while the most efficient A-star h2 only solves 38.95% of solvable 8-puzzles.
    - At condition of maxNodes upper limit 5000, even the slowest local beam search with k =100 solves 100% of solvable 8-puzzles, while h2 solves 97.94% and h1 solves only 36.81%.

> (b) For A* search, which heuristic is better?

No matter following the criteria of time consumed, space efficiency, or solvability, h2, which is the Manhattan distance is far superior than h1, which is the number of misplaced tiles.

- Under maxNodes upper limit 1000, h2 heuristic solves approximately 6 times more solvable 8-puzzle than h1 heuristic does. At condition of maxNodes upper limit 5000, h1 heuristic finds the optimal path of 97.94% of all solvable 8-puzzle games, while h1 only gets 36.81%. Even at condition of maxNodes upper limit 10000, h1 only gets 53.64% solved.
- Time consumed for h1 is way higher than h2. Even for the simplest questions in all the solvable states, at condition of maxNodes upper limit 100, h1 is 10 times slower than h2 in the term of average time consumed per question. For the most difficult states that might requires 25+ moves, h1 fails miserably: at condition of maxNodes upper limit 10000, h1 is 27 times slower than h2 in the term of average time consumed per question, given the fact that h2 solves all states and h1 only solves easier half of all the states.
- Average nodes explored per puzzle state by h1 is much more than h2. Even h2 solves more puzzle states under the same maxNodes condition, which means higher difficulty in average, the nodes explored by h1 is similar to h2 under the condition of maxNodes upper limit less than 1000. When the problems get more difficult, h1 explores twice more nodes at the condition of maxNode 5000 and four times more nodes at the condition of maxNode 10000 than h2.

> (c) How does the solution length vary across the three search methods?

Given the nature of A-star search and local beam search, the solution provided by no matter h1 and h2 will be complete and optimal; while since my adapted A-star search settled down at whatever path to goal state it finds, it is not necessarily optimal at most cases. One general rule for local beam search is that more beams we use, more closer to the optimal path length for the results. For the sake of completeness, we will focus on condition of maxNodes upper limit 10000.

- If h1 and h2 could both finish all the 8-puzzle states, the path length will be identical. In this case, h1 does not finish the more complicated half of solvable states, and therefore yields shorter average path length.
- Local beam search with fewer beams will be faster, but the resulting path length will be longer; vice versa.

Beam Number	Average Path Length for All Solvable States

Beam Number	Average Path Length for All Solvable States
2	89
3	72
5	58
10	42
20	34
30	31
50	29
100	27

> (d) For each of the three search methods, what fraction of your generated problems were solvable?

As mentioned before, local beam k = 2 solves 100% within 500 maxNodes, local beam k =100 solves 100% within 5000 maxNodes, A-star h2 solves 100% within 10000 maxNodes. Therefore, it is confident to claim that for local beam and A-star h2, we could solve all solvable 8-puzzle states within 10000 maxNodes, which is the default setting for my program. However, the time required for A-star h1 grows exponentially when solving the more complicated half of all solvable 8-puzzle states, and as as result, only solves roughly half of all solvable state for 8-puzzle under the default setting.

---

4. Discussion

> (a) Based on your experiments, which search algorithm is better suited for this problem? Which finds shorter solutions? Which algorithm seems superior in terms of time and space?

- If in term of optimality, A-star algorithm is obviously the go-to choice for 8-puzzle game. As asked in the second questions, we can refer to the table in Experiment section c part, which we could see even with beam number 100, the average path length is only 27 for all solvable path; while the optimal paths found by h2 shows the average is approximately 22 for all solvable path. The gap cannot be easily crossed by simply adding the number of beams: more beams means worse performance, even worse than A-star h2 performance. And therefore, A-star h2 finds the shorter solutions for all solvable states, and thus in this sense, the best choice.
- If in term of time and space, local beam search with smallest possible beam amount will be the choice. Although in the table mentioned before, the local beam search with k = 2 has average path length of 89 for all solvable states, the path consideration is out of scope. Instead, for all solvable states, the average nodes explored for local beam k = 2 is only 167, compared to h2's 1945 nods explored; meanwhile, the time consumed to solve all states for local beam k = 2 is 41092 ms, which is 41 seconds, compared to 2290539 ms, or approximately 38 minutes that h2 spent. Therefore, local beam search with k = 2 is the most superior choice in this sense.
- Also for the local beam with larger number of beams, like 20, 30, or 50 beams, the performance is still significantly better than A-star h2, while the average path length is around 25 - 30 steps for all solvable states. These will be good choices for the users who deserves both aspects.

> (b) Discuss any other observations you made, such as the difficulty of implementing and testing each of these algorithms.

During the implementation of adapted local beam search, the first version was implemented by the same idea: (preferentially put unexplored successors with highest h2 value into bestkNodes.)

- First, Initialize all local variable. The heuristic function used in this algorithm is h2, sum of the distance of the tiles from their goal positions.
- Second, put the starting state and numberOfState - 1 states that randomly move 10 steps from the starting state. These states fill the bestkNodes, which is the ArrayList that stores the k nodes with smallest heuristic value.
- Third, until we find the goal state or reach the upper limitation of node exploration, we put all the successors of those states in the bestkNodes into a temporary PriorityQueue. After finding all successors, we clear bestkNodes list and put the top k nodes within priorityQueue into bestKNodes list.

However, the observation made by running this local beam search algorithm is that for some states, especially the more complicated ones, the algorithm will run for a few minutes and generate hundreds of thousands of successor generations. In these case, the algorithm is even slower than brutal force algorithm. So I printed the best k nodes each generations out and tried to figure out what is going wrong.

It turns out that the algorithm is likely to hold local maximums in the best k lists and get in an infinite loop. So I added the ArrayList just like what I did in A-star algorithm to store the explored State's hash code and preferentially put the unexplored nodes into bestkNodes list and avoids the repeated nodes because they are highly likely to be local maximums unless the bestkNodes list is not full. The change turns out to be successful, and the local beam search turns out to behave as expectation: fast algorithm that finds a path to goal state, not necessarily complete.