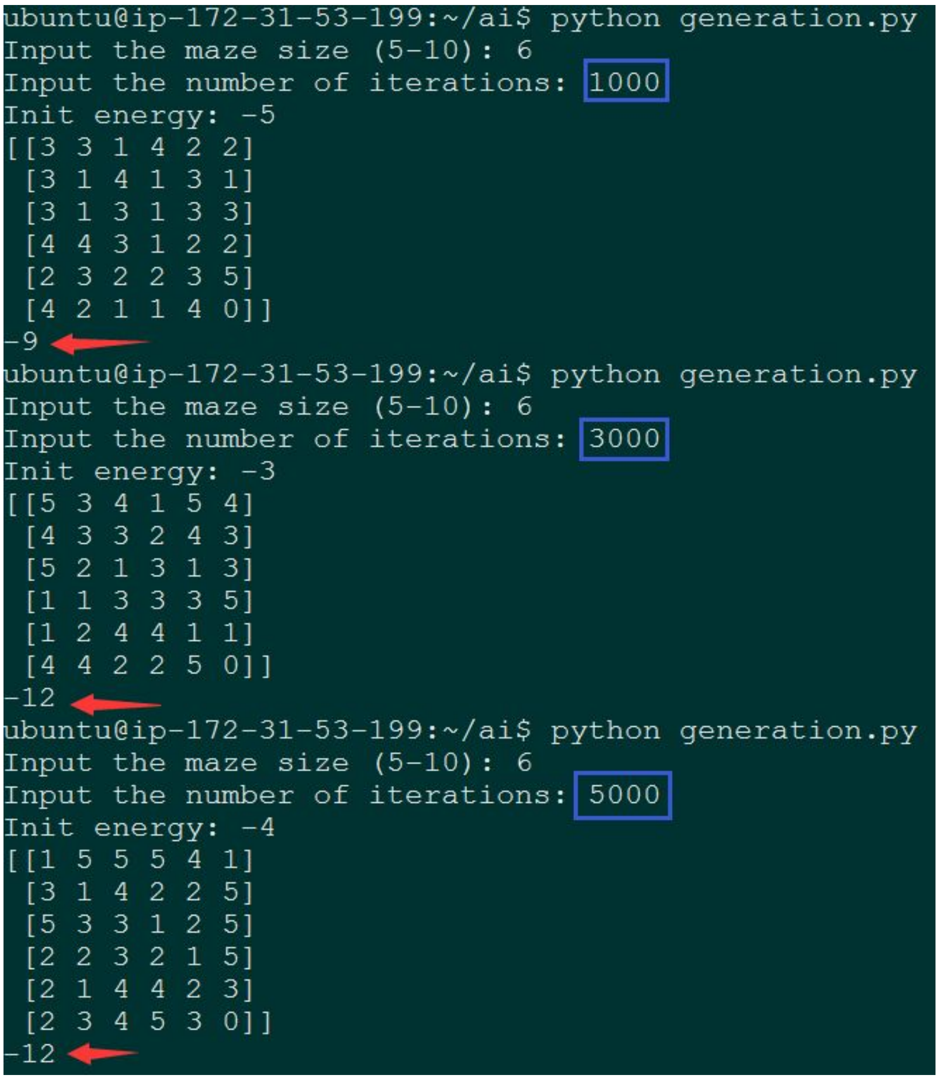
Problem 3 Questions:

1. *Empirically, how does the objective function evaluation of the average generated maze vary with the number of iterations?*

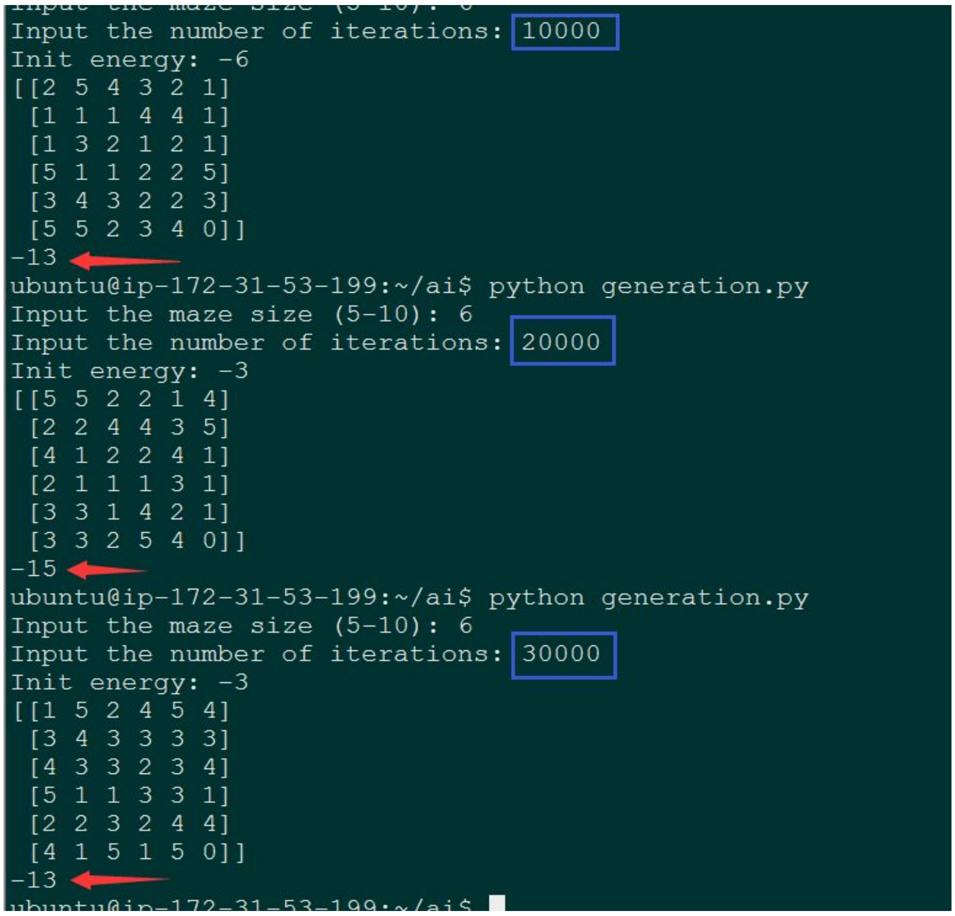
2. *Why should we allow sideways moves (objective function is the same)? That is, how might sideways moves help stochastic local search?*

1. As the number of iterations grows, the average energy of the maze are getting lower:



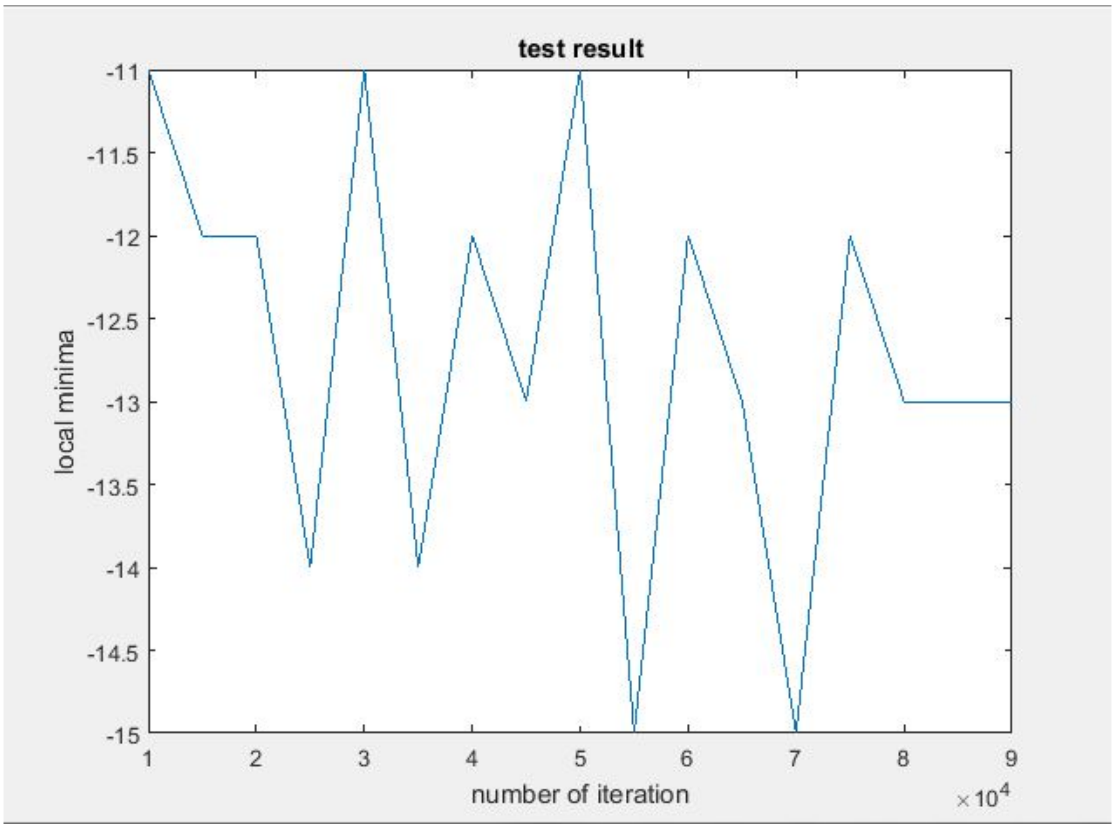
```
ubuntu@ip-172-31-53-199:~/ai$ python generation.py
Input the maze size (5-10): 6
Input the number of iterations: 1000
Init energy: -5
[[3 3 1 4 2 2]
 [3 1 4 1 3 1]
 [3 1 3 1 3 3]
 [4 4 3 1 2 2]
 [2 3 2 2 3 5]
 [4 2 1 1 4 0]]
-9
ubuntu@ip-172-31-53-199:~/ai$ python generation.py
Input the maze size (5-10): 6
Input the number of iterations: 3000
Init energy: -3
[[5 3 4 1 5 4]
 [4 3 3 2 4 3]
 [5 2 1 3 1 3]
 [1 1 3 3 3 5]
 [1 2 4 4 1 1]
 [4 4 2 2 5 0]]
-12
ubuntu@ip-172-31-53-199:~/ai$ python generation.py
Input the maze size (5-10): 6
Input the number of iterations: 5000
Init energy: -4
[[1 5 5 5 4 1]
 [3 1 4 2 2 5]
 [5 3 3 1 2 5]
 [2 2 3 2 1 5]
 [2 1 4 4 2 3]
 [2 3 4 5 3 0]]
-12
```

However, this trend is not keep growing as the the number reach a certain threshold due to it stuck in the local minima:



```
Input the maze size (5-10): 6
Input the number of iterations: 10000
Init energy: -6
[[2 5 4 3 2 1]
 [1 1 1 4 4 1]
 [1 3 2 1 2 1]
 [5 1 1 2 2 5]
 [3 4 3 2 2 3]
 [5 5 2 3 4 0]]
-13
ubuntu@ip-172-31-53-199:~/ai$ python generation.py
Input the maze size (5-10): 6
Input the number of iterations: 20000
Init energy: -3
[[5 5 2 2 1 4]
 [2 2 4 4 3 5]
 [4 1 2 2 4 1]
 [2 1 1 1 3 1]
 [3 3 1 4 2 1]
 [3 3 2 5 4 0]]
-15
ubuntu@ip-172-31-53-199:~/ai$ python generation.py
Input the maze size (5-10): 6
Input the number of iterations: 30000
Init energy: -3
[[1 5 2 4 5 4]
 [3 4 3 3 3 3]
 [4 3 3 2 3 4]
 [5 1 1 3 3 1]
 [2 2 3 2 4 4]
 [4 1 5 1 5 0]]
-13
ubuntu@ip-172-31-53-199:~/ai$
```

We can see when the iteration number is 30,000, the energy is worse than 20,000. It might be fall into local minima.



10000~90000(every 5000)

2 As long as sideways moves can improve the generated energy, it might generate a smaller energy than the current local minima with certain probability. That's why we call this algorithm stochastic search algorithm.

There are so many paths to the final best maze, the search space is sort of infinite as the maze grows. In this method, we evaluate our next move based on current location. Sideways moves generate another reasonable states.

Problem 4 Questions:

1. *Empirically, keeping the total number of iterations (i.e. descents times iterations per descent) constant (e.g. 10,000), approximately how many restarts yields the best average generated maze?*

2. *For which kind of optimization tasks is it better not to restart search? That is, for which kinds of functions or iteration constraints is it better to do a single descent for all iterations?*

1 It's hard to say. In my test, 6 restarts gain the best result:

```
ubuntu@ip-172-31-53-199:~/ai$ python restart.py
Input the number of restarts: 6
Input the maze size (5-10): 6
Input the iteration number per restart: 10000
minimum energy: -19
best maze:
[[4 1 5 5 2 3]
  [3 4 3 2 1 3]
  [5 4 2 1 1 3]
  [4 2 1 2 1 2]
  [2 1 2 1 4 2]
  [1 2 1 3 4 0]]
```

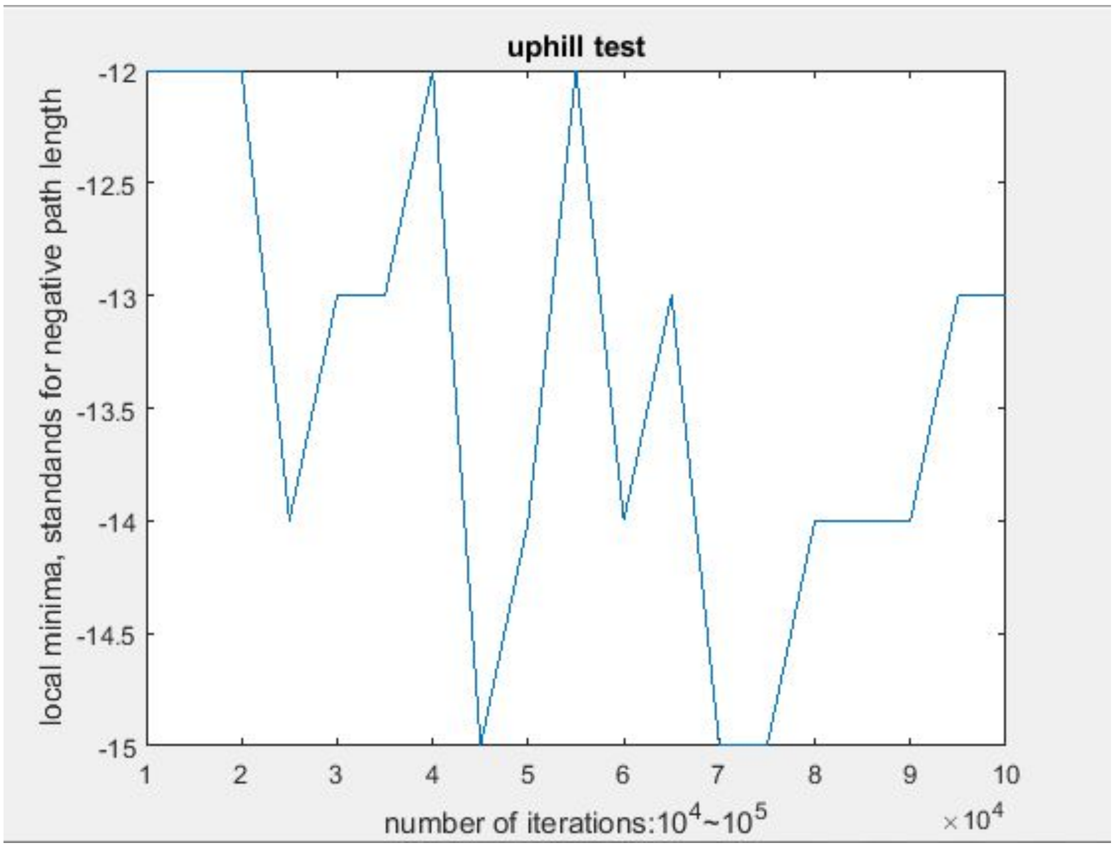
2 It depends on the shape of the state-space. If there's only one global maxima and without local maxima, it's not good to restart. On the contrary, use all the iterations from an randomly initial state is much likely to achieve the best result.

Problem 5 Questions:

1. *Empirically, how does the objective function evaluation of the average generated maze vary with (1) the number of iterations, and (2) the probability of accepting uphill steps?*
2. *Is the probability of taking an uphill step the same as the probability of escaping a local minimum? Why or why not?*

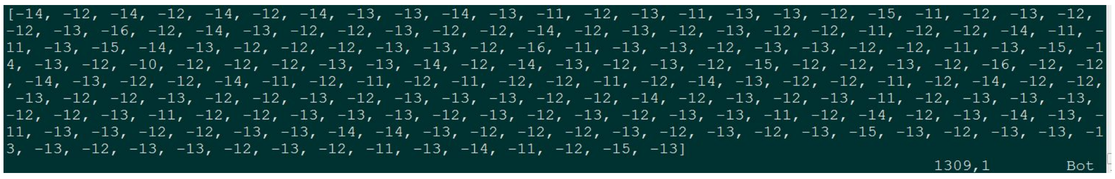
1 For the number of iteration(20 iterations, 5*5 maze):

```
y =
-12     -12     -12     -14     -13     -13     -12     -15     -14     -12     -14
-13     -15     -15     -14     -14     -14     -13     -13
>> mean(y)
ans =
   -13.3684
```



2 I run 200 iterations, the probability ranges from 0.0001~0.1

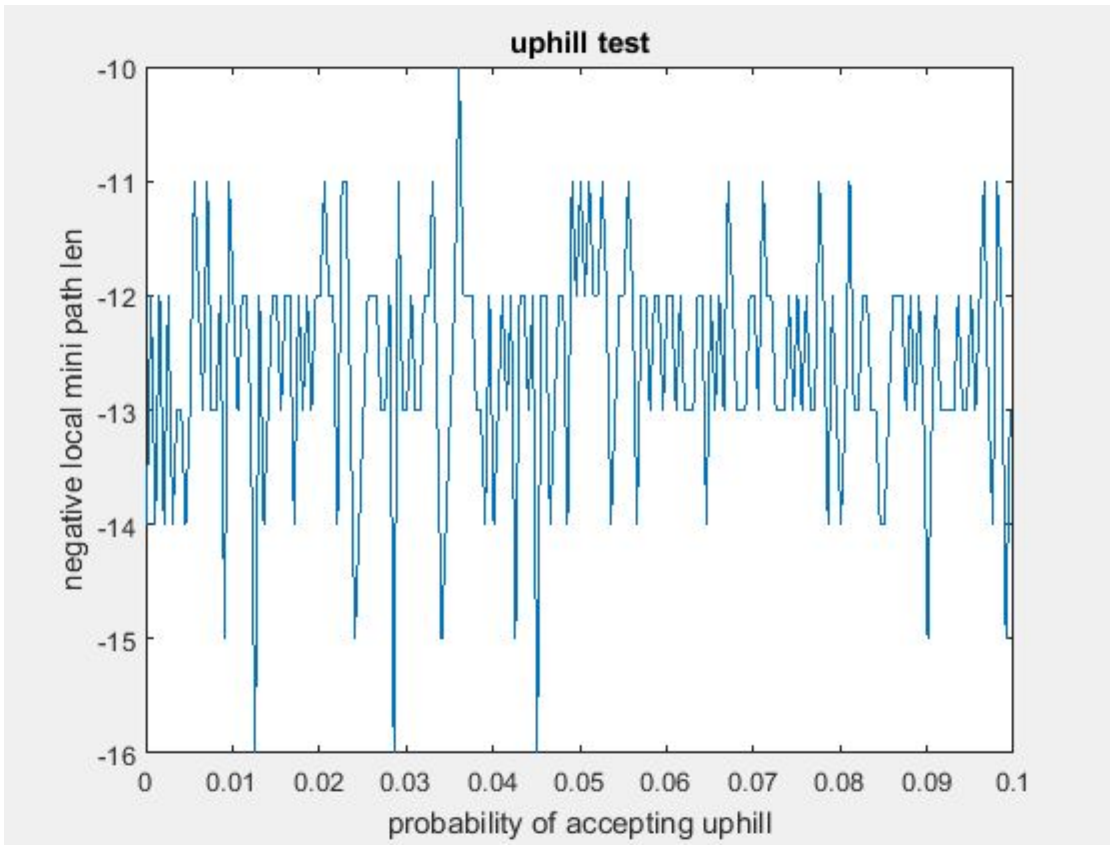
The negative path length:

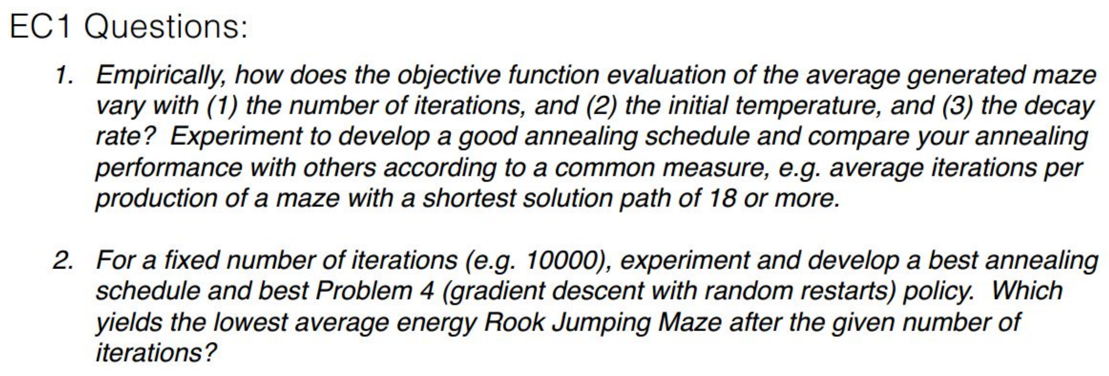


```
[-14, -12, -14, -12, -14, -12, -14, -13, -13, -14, -13, -11, -12, -13, -11, -13, -13, -12, -15, -11, -12, -13, -12,
-12, -13, -16, -12, -14, -13, -12, -12, -13, -12, -12, -14, -12, -13, -12, -13, -12, -12, -11, -12, -12, -14, -11, -
11, -13, -15, -14, -13, -12, -12, -12, -13, -13, -12, -16, -11, -13, -13, -12, -13, -13, -12, -12, -11, -13, -15, -1
4, -13, -12, -10, -12, -12, -12, -13, -13, -14, -12, -14, -13, -12, -13, -12, -15, -12, -12, -13, -12, -16, -12, -12
, -14, -13, -12, -12, -14, -11, -12, -11, -12, -11, -12, -12, -11, -12, -14, -13, -12, -12, -11, -12, -14, -12, -12,
 -13, -12, -12, -13, -12, -12, -13, -12, -13, -13, -13, -12, -12, -14, -12, -13, -12, -13, -11, -12, -13, -13, -13,
-12, -12, -13, -11, -12, -12, -13, -13, -13, -12, -13, -12, -13, -12, -13, -12, -13, -13, -11, -12, -14, -12, -13, -14, -13, -
11, -13, -13, -12, -12, -13, -13, -14, -14, -13, -12, -12, -12, -13, -12, -13, -12, -13, -15, -13, -12, -13, -13, -1
3, -13, -12, -13, -13, -12, -13, -12, -11, -13, -14, -11, -12, -15, -13]
                                                                                        1309,1          Bot
```

The mean of the path:

mean(b)=-12.6050





EC1 Questions:

1. *Empirically, how does the objective function evaluation of the average generated maze vary with (1) the number of iterations, and (2) the initial temperature, and (3) the decay rate? Experiment to develop a good annealing schedule and compare your annealing performance with others according to a common measure, e.g. average iterations per production of a maze with a shortest solution path of 18 or more.*

2. *For a fixed number of iterations (e.g. 10000), experiment and develop a best annealing schedule and best Problem 4 (gradient descent with random restarts) policy. Which yields the lowest average energy Rook Jumping Maze after the given number of iterations?*

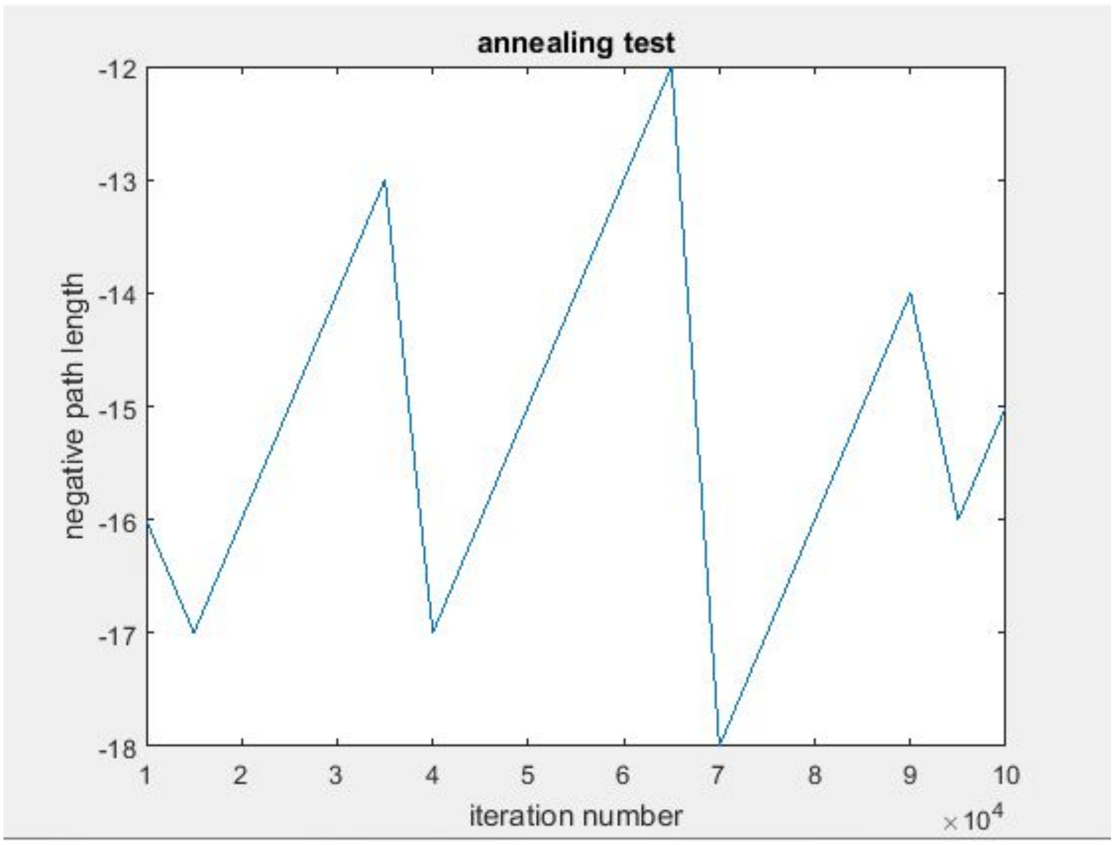
1

The number of iterations:

According to my experiments, it’s likely to have a good minima as the number increasing. However, it doesn’t guarantee this conclusion.

```
x1 =
   Columns 1 through 16
          10000            15000            20000            25000
30000         35000         40000         45000         50000
55000         60000         65000         70000         75000
80000         85000
   Columns 17 through 19
          90000         95000         100000

>> x2
x2 =
    -16    -17    -16    -15    -14    -13    -17    -16    -15    -14
-13    -12    -18    -17    -16    -15    -14    -16    -15
>> mean(x2)
ans =
   -15.2105
```



The initial temperature:
The best temperature I test is 2.5
If I change temperature to another value, the results turn out very bad.

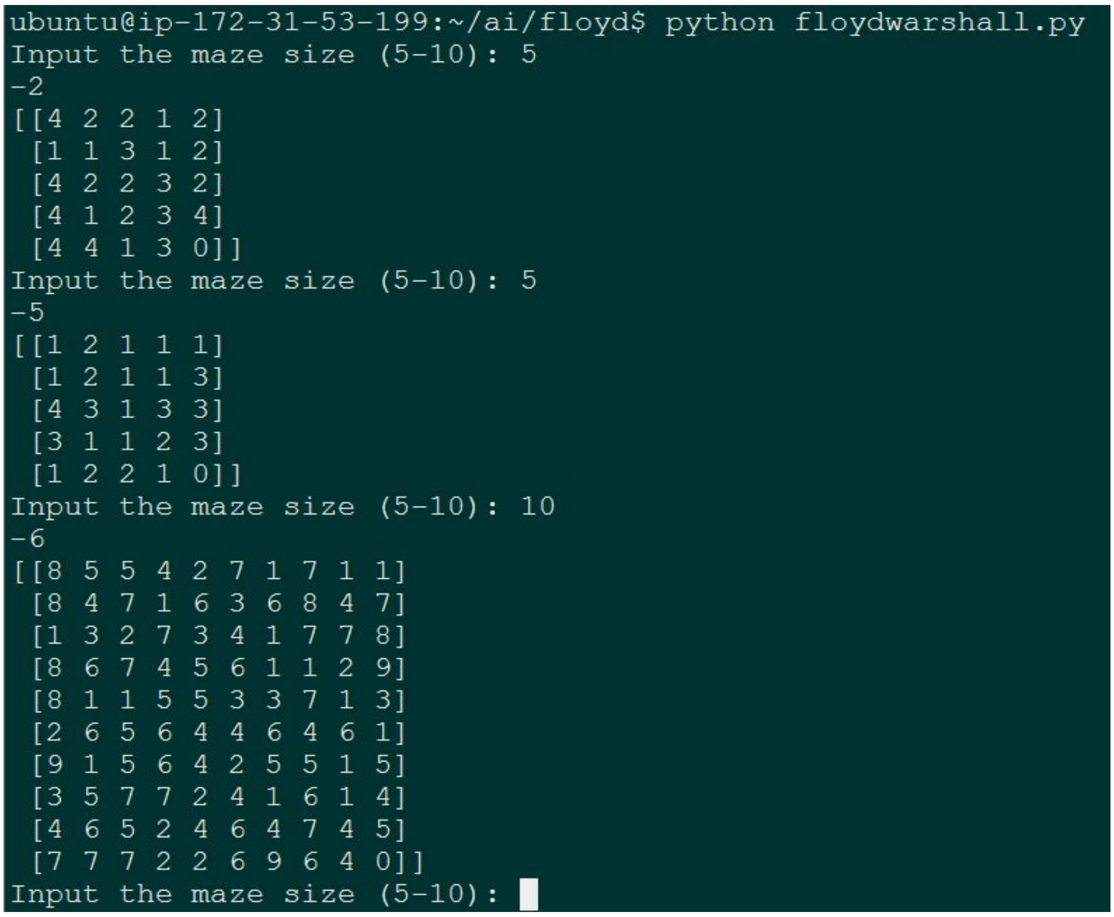
The decay rate:
The best decay rate is 0.99999
Due to the iteration number is relatively large, if the decay rate is pretty sensitive, a little more higher will cause it to 0 in an very early phase.

2 In my test, the annealing yield best result, which is -18 while in the random restart is about -15 with 10000 iterations.

EC2

Select one or more of the features above that you consider most important, compute measures for maze evaluation, and seek to improve upon the previous RJM evaluation function. Generate sets of puzzles using the old and new evaluation functions, compare and contrast the puzzles, and argue why your new evaluation function improves the average quality of your mazes.

I change the evaluation function to floyd-warshall algorithm to calculate the distance between the start state and goal state:
The mazes generated using the new method:



```
ubuntu@ip-172-31-53-199:~/ai/floyd$ python floydwarshall.py
Input the maze size (5-10): 5
-2
[[4 2 2 1 2]
 [1 1 3 1 2]
 [4 2 2 3 2]
 [4 1 2 3 4]
 [4 4 1 3 0]]
Input the maze size (5-10): 5
-5
[[1 2 1 1 1]
 [1 2 1 1 3]
 [4 3 1 3 3]
 [3 1 1 2 3]
 [1 2 2 1 0]]
Input the maze size (5-10): 10
-6
[[8 5 5 4 2 7 1 7 1 1]
 [8 4 7 1 6 3 6 8 4 7]
 [1 3 2 7 3 4 1 7 7 8]
 [8 6 7 4 5 6 1 1 2 9]
 [8 1 1 5 5 3 3 7 1 3]
 [2 6 5 6 4 4 6 4 6 1]
 [9 1 5 6 4 2 5 5 1 5]
 [3 5 7 7 2 4 1 6 1 4]
 [4 6 5 2 4 6 4 7 4 5]
 [7 7 7 2 2 6 9 6 4 0]]
Input the maze size (5-10):
```

The floyd-warshall algorithm can get distance in O(n^3) running time, when the maze size get bigger, it's more efficient than BFS. More details please refer to the cods in the attachments.