**CS 520: Assignment 1**

|  |  |
| --- | --- |
| Yan Gu | yg369 |
|  |  |
|  |  |
|  |  |

Group members:

1. **Part 1:**

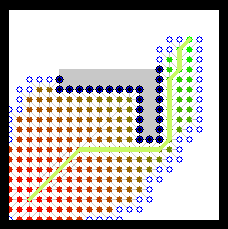
**Generating Environments:**

**DFS:**

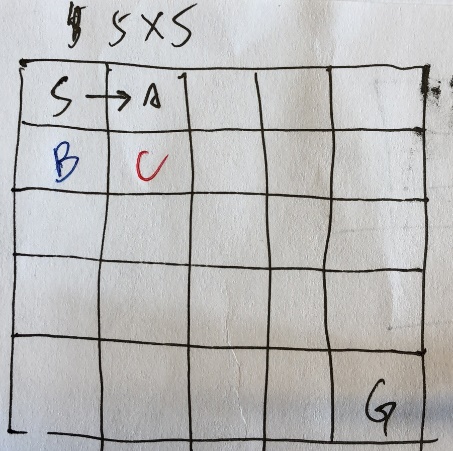
**BFS:**

***A\** :**

The picture shows a classic example of A\* ALG. I use a closed set to maintain the nodes we have already traveled. And when we arrive a node, we will discover its neighbor and I store them in a min-heap called *open set* so we can always pop the neighbor with minimum **f.**



But I found that there may be a tie when there are some nodes in the *open set* have same **f** value, by default, we will pop the first node in the *open set*. For example, using the Manhattan Distance, the following picture states a tie. In a 5\*5 grid, when we arrive node *A*, we can easily know that node *B* and *C* is the first and second node in the *open set*, but **f**B = 1+(5-1)+(5-2) = 8, **f**C = 2+(5-2)+(5-2) = 8.



Obviously, C is better than B, but B may be pop in front of C, because we can never directly know which node is better in a larger grid, so we need a tie breaker. I choose a strategy that when several nodes have same **f**, we always choose a node with higher **g** value since we think it may be closer to the goal. Using this tie-breaking method, we can expand fewer nodes than usual.

Input: numpy.array

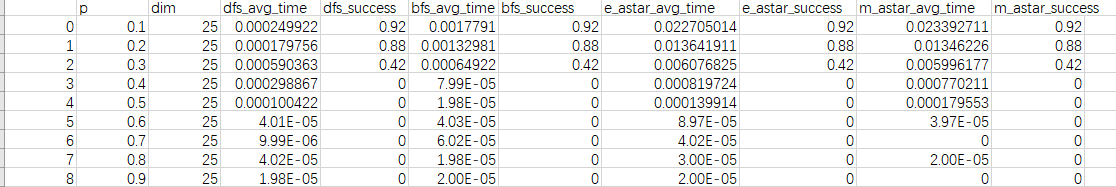
Output: dictionary(path, nodes\_number, fringe)

**Question answer:**

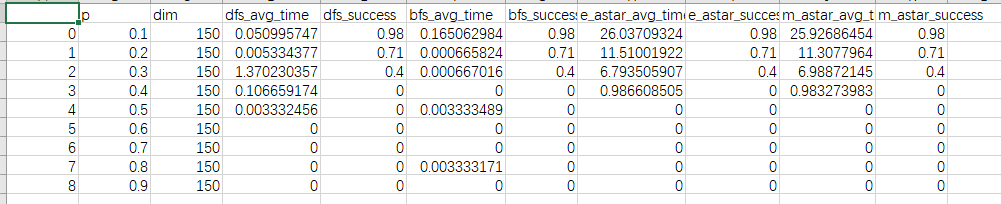
1. We test for each p = 0.1, 0.2, …, 0.9, and in each step, we test for dim = 20, 25, 30, …, 200. We found that when p = 0.1, the time cost is biggest compared to other p value. And the probability for finding a path is almost the same between different dim. I put some of the test data in the submission. \_avg\_time is the time for each finding process in seconds. \_success is the probability for succeeding in finding a path.

Here are some examples, based on them, we choose dim = 160 for the following test of **Part 1**.

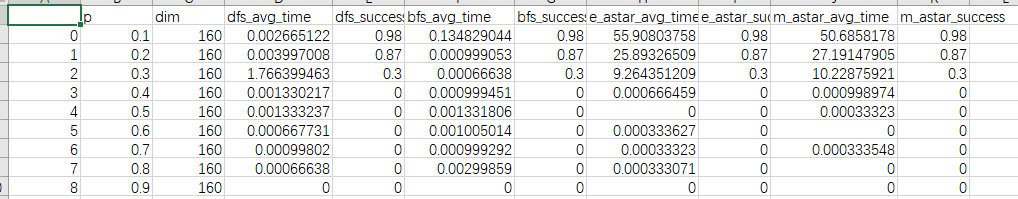
**Dim = 25**



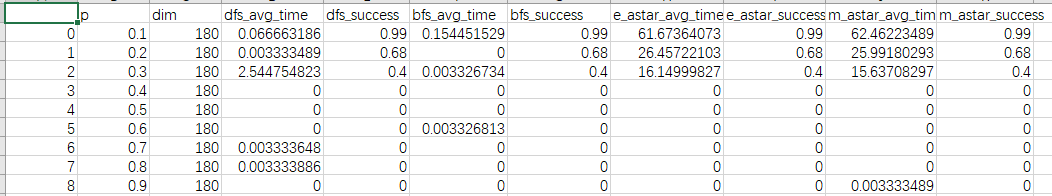
**Dim = 150**



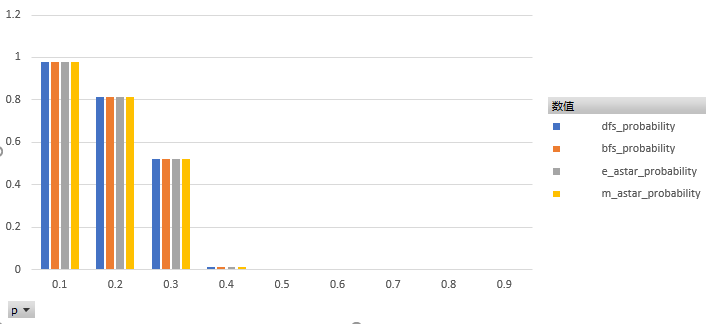
**Dim = 160**

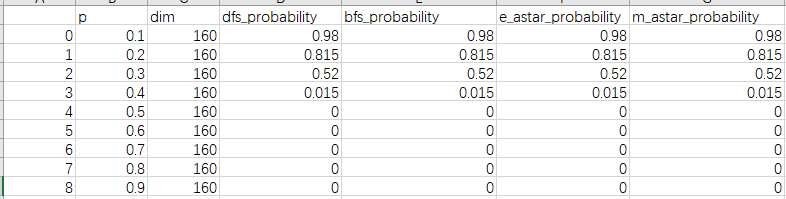


**Dim = 180**



1. For each p, we use 1000 random mazes to test, the probability for finding a path shows below.

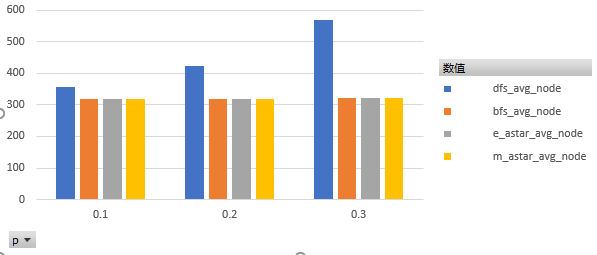


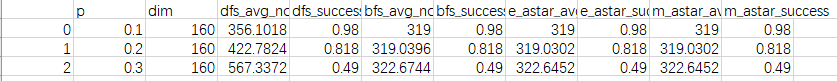


So for all four ALG, we choose p0 = 0.3

1. We test 1000 random mazes, from the data we can easily know that the BFS and two A\* have the same shortest path and the expected length of the shortest path for DFS is larger than the other three. And BFS is slightly larger than two A\*. So in this question, we can conclude that the two A\* are the most useful ALG.

\_avg\_node represents the average length of path, \_success represents the probability of finding a path.





1. Use the data we already have in Question 4, the length of path generated by DFS is obviously larger than two A\*, and the lengths of path generated by two A\* are the same.

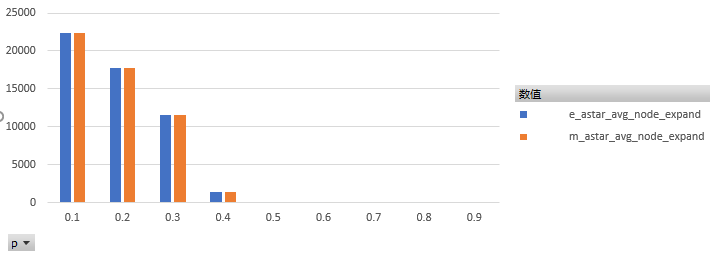
And as the p getting bigger, the gap between DFS and A\* is getting much bigger.

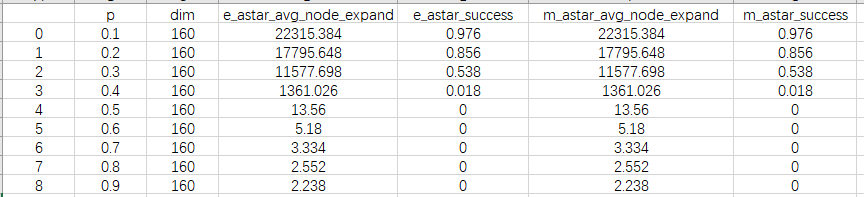
1. Because I use a tie-breaking strategy beforehand, so the total number of expanded nodes will be the same for either heuristic since the influence caused by different heuristic in choosing next step is dropped off by the tie breaker. The tie breaker will force two algorithm to do the same *right* thing, that is, to get closer to the goal as far as possible.

So for all p, there is no difference.

\_avg\_node\_expand represents the average number of total nodes expanded.

\_success represents the probability of finding a path.



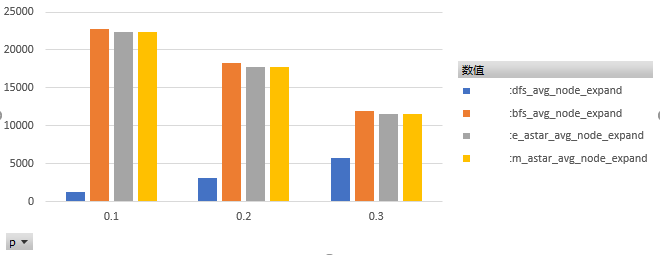


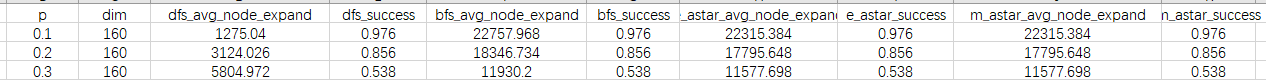
1. The DFS expanded the fewer nodes than other three, because it finds the first path from start to the goal regardless whether the path is the shortest. But BFS and two A\* aim to find the shortest path, so they will traverse a larger part of grid than DFS.

Data shows below.

\_avg\_node\_expand represents the average number of total nodes expanded.

\_success represents the probability of finding a path.





Bonus 1)