A close up of a piece of paper

Description automatically generated

**Part 2**

**4)**

**For p in [0, p0] as above, estimate the average or expected length of the shortest path from start to goal. You may discard unsolvable maps. Plot density vs expected shortest path length. What algorithm is most useful here?**

**Compare 2 different A\* algorithm:**

|  |  |  |  |
| --- | --- | --- | --- |
| Euclidean\_Distance dim=512 |  |  |  |
| p | path\_length | p | path\_length |
| 0 | 1023 | 0.09 | 1025 |
| 0.01 | 1023 | 0.1 | 1023 |
| 0.02 | 1023 | 0.12 | 1023 |
| 0.03 | 1023 | 0.13 | 1031 |
| 0.04 | 1023 | 0.14 | 1035 |
| 0.05 | 1023 | 0.15 | 1029 |
| 0.06 | 1023 | 0.16 | 1037 |
| 0.07 | 1023 | 0.17 | 1039 |
| 0.08 | 1023 | 0.18 | 1039 |

A screenshot of a social media post

Description automatically generated

|  |  |  |  |
| --- | --- | --- | --- |
| Manhattan\_Distance dim=512 |  |  |  |
| p | path\_length | p | path\_length |
| 0 | 1023 | 0.1 | 1023 |
| 0.01 | 1023 | 0.11 | 1023 |
| 0.02 | 1023 | 0.12 | 1023 |
| 0.03 | 1023 | 0.13 | 1023 |
| 0.04 | 1023 | 0.14 | 1023 |
| 0.05 | 1023 | 0.15 | 1023 |
| 0.06 | 1023 | 0.16 | 1023 |
| 0.07 | 1023 | 0.17 | 1023 |
| 0.08 | 1023 | 0.18 | 1023 |
| 0.09 | 1023 | 0.19 | 1023 |

A screenshot of a social media post

Description automatically generated

Cause Bi\_Directional BFS cost lots of time to run dim=512 we use dim=100

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Bi\_Directional\_Breadth\_First\_Search dim-100 |  |  |  |  |  |
| p | meeting\_node | path\_length | p | meeting\_node | path\_length |
| 0 | (50,49) | 199 | 0.1 | (33, 66) | 199 |
| 0.01 | (45, 54) | 199 | 0.11 | (40, 59) | 199 |
| 0.02 | (46, 53) | 199 | 0.12 | (72, 27) | 199 |
| 0.03 | (49, 50) | 199 | 0.13 | (74, 25) | 199 |
| 0.04 | (46, 53) | 199 | 0.14 | (61, 38) | 199 |
| 0.05 | (45, 54) | 199 | 0.15 | (72, 27) | 199 |
| 0.06 | (37, 62) | 199 | 0.16 | (24, 75) | 199 |
| 0.07 | (20, 79) | 199 | 0.17 | (42, 58) | 199 |
| 0.08 | (79, 20) | 199 | 0.18 | (29, 71) | 199 |
| 0.09 | (44, 55) | 199 | 0.19 | (60, 38) | 199 |

A screenshot of a social media post

Description automatically generated

**Above We can conclude that BiDirectional BFS is time-wasting and A\* algorithm perform better**

**5)**

**Is one heuristic uniformly better than the other for running A∗ ? How can they be compared? Plot the relevant data and justify your conclusions.**

I develop UniformSearch Algorithm, supposing heuristic is 0 and compare with A\* algorithm

**compare three different algorithm time cost:**

set dim=512 and p is ranged from 0 to 0.6 with step 0.05:

total time cost using Manhattan search is 8.99621868133545

total time cost using Euclidean search is 9.80755090713501

total time cost using Uniform search is 11.38677978515625

set dim=512 and p is ranged from 0 to 0.3 with step 0.025:

total time cost using Manhattan search is 12.999441146850586

total time cost using Euclidean search is 12.308625221252441

total time cost using Uniform search is 16.55923104286194

set dim=512 and p is ranged from 0 to 0.2 with step 0.02:

total time cost using Manhattan search is 12.800921201705933

total time cost using Euclidean search is 14.627891063690186

total time cost using Uniform search is 16.26413106918335

Dijkstra is a special case for A\* (when the heuristics is zero)

From the data we can conclude that uniformsearch(heuristic uniformly) spend more time than A\* algorithm

**6) Do these algorithms behave as they should?**

Yes BdBFS cost lots of time and A\* algorithms perform better

**8) On the same map, are there ever nodes that BD-DFS expands that A∗ doesn’t? Why or why not? Give an example, and justify.**

Yes Different algorithm may generate different path and the nodes visited is different.

Using copy.deepcopy library to duplicate 3 same mazes and use 3 algorithms to calculate the sum of visited node

Here the snapshot of result:

Dim=256 p=0.2 (Here 100 represent path)

Original Path

A black sign with white text

Description automatically generated

For Mahattam A\*

A close up of a keyboard

Description automatically generated

For Euclidean A\*

A close up of a keyboard

Description automatically generated

For Bd-BFS:

A picture containing text

Description automatically generated

**Part 3**

**a)What local search algorithm did you pick, and why? How are you representing the maze/environment to be able to utilize this search algorithm? What design choices did you have to make to make to apply this search algorithm to this problem?**

Using simulated annealing algorithm, which is better than climbing hill algorithm and we can jump out of the local optimal solution. I think there is no regular to follow so it is not proper to use generic algorithm.

We use ndarray library to represent maze, you can also treat it as 2d list. We can print out the solved maze:0 represent empty 1 represent filled and 100 represent path. We will input a. solvable maze and find the hardest maze using the designed algorithm

Because for different algorithms we need different parameter returned so I wrapped the Astar Manhatan and DFS as classes(we have 2 Astar Manhattan and 2 DFS python files, the files named ended with “class” are applied in part 3 and the others are applied in part 2 and1)

We build our simulated annealing algorithm based on the class notes and modified a little bit to fit our requirement, We mainly use two parameter(get\_node\_visited,get\_path\_length) imported by Astar and DFS to get the maximum value using local search algorithm

The input maze size=10 and p=2

For generate random neighbor maze I will still keep the same property (once we remove a wall cell we will break a wall but keep the maze solvable)

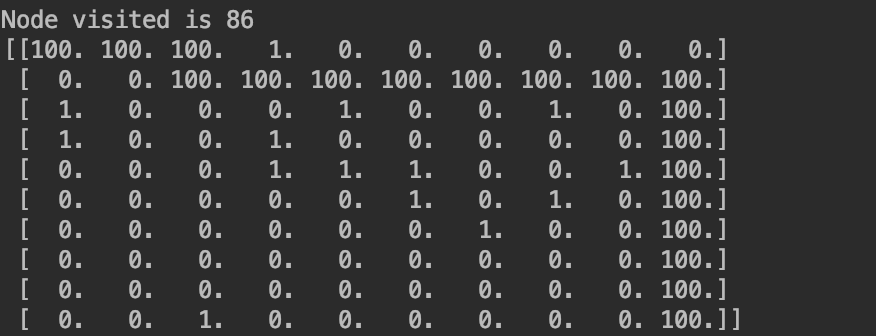
**b)Unlike the problem of solving the maze, for which the ‘goal’ is well-defined, it is difficult to know if you have constructed the ‘hardest’ maze. What kind of termination conditions can you apply here to generate hard if not the hardest maze? What kind of shortcomings or advantages do you anticipate from your approach?**

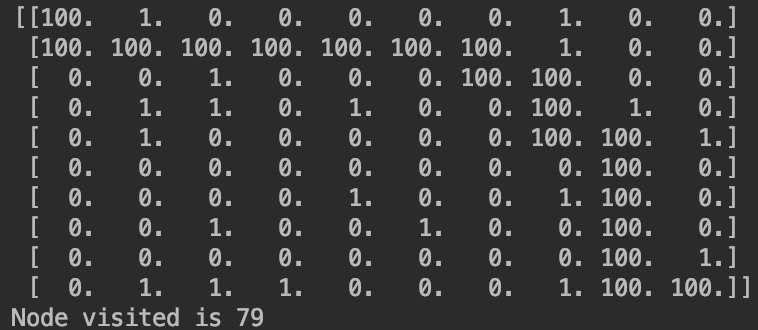
Because I use simulated annealing algorithm so once the temperature cold down and the state is stable, I will return the final result. But how can we measure the stable state? My algorithm designs a termination condition to value if the value is stable. We will keep record the time the maze enters into stable state. If the temperature cold down and the maze continuously enter into the stable state for 200 times, we will confirm that this is the final result. We can change the threshold, but I think for a maze with dim=10 and p=0.2 which is proper

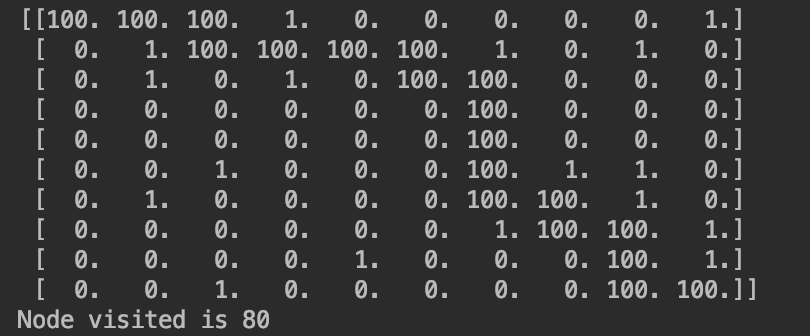
I think my algorithm’s advantage is I can set the precision threshold for my final result by changing a certain parameter. The outcoming is that I cannot guarantee that the result is the global maximum value each time. I think it is the universal shortcoming for evert local search algorithm

**c)I will attach my result and relevant date below:**

For Astar Manhhatam(simply run the python file” Simulated\_annealing”)







For DFS (simply run the python file” Simulated\_annealing”)

A close up of a keyboard

Description automatically generated

A picture containing keyboard

Description automatically generated

A close up of a keyboard

Description automatically generated

**d)yes, Although there are a little bit different among 3 times running result but we still can get a harder maze**