CS 520: Assignment 1 - Path Planning and Search Algorithms

Shengjie Li

October 7, 2018

1 Introduction, group members and division of workload

In this group project, apart from implementing DFS, BFS, A^* with Euclidean Distance and A^* with Manhattan Distance, we also did some modification to these algorithms for different performance out of our personal interests.

Name RUID	Workload
Haoyang Zhang 188008687	Implemented DFS, Iterative Deepening DFS, BFS, Bidirectional BFS, Bidirectional A^*, Beam Search, Simulated Annealing and the visualization of maze. Modified DFS to make it able to return optimal path. Added Last-in First-out feature to A^*. Managed to combine Beam Search, Simulated Annealing and Genetic Algorithm. Ran tests for DFS and BFS in question 10. Finished half of the writing of report for part 2.
Han Wu 189008460	Wrote python scripts for testing the performance of algorithms. Combine the data and generated figures for question 1, 2, 4 and 5. Finished the writing of report for question 1, 2, 4 and 5.
Shengjie Li 188008047	Implemented A^* with Euclidean Distance, A^* with Manhattan Distance and Genetic Algorithm. Ran tests for A^* with Euclidean Distance and Manhattan Distance inquestion 10. Finished the format design of whole report.
Zhichao Xu 188008912	Wrote python scripts for testing the performance of algorithms. Combine the data and generated figures for question 3, 6 and 7. Finished the writing of report for question 3, 6 and 7. Suggested an improvement of A^* using Chebyshev Distance.

2 Part 1: Path Planning

1 For each of the implemented algorithms, how large the maps can be (in terms of dim) for the algorithm to return an answer in a reasonable amount of time (less than a minute) for a range of possible p values? Select a size so that running the algorithms multiple times will not be a huge time commitment, but that the maps are large enough to be interesting.

In order to find a reasonable size of the maze, we tested the running time of the algorithm. For each size, we generated 10 mazes and run different algorithms on these 10 mazes. We recorded the average time each algorithm needs to return an answer. The 10 mazes in each test ccould be either solvable or unsolvable. We set p equals to 0.3 and executed the

experiment.
The results are shown below as Table 1:

		Size					
		100	200	400	800	1600	3200
Time(s)	DFS	0.01942	0.07087	0.28435	0.93919	4.5064	10.83366
	BFS[2]	0.07342	0.33946	1.73646	6.40967	25.69253	91.73234
	A^* Euclidean	0.07811	0.41706	1.62604	5.97804	25.27724	100.37974
	A^* Manhattan	0.06093	0.29222	0.88752	3.63068	11.147	63.8882
	BFS[1]	254.36093 (size=30)					

Table 1

DFS was the default setting. BFS1 was the default setting. However, it took too much time. When the size was 30, it took more than 250 seconds to return an answer. So, we changed the settings a little to make BFS acceptable. Here came BFS2. In BFS2, checkFringe=True. The others were also False. A* Euclidean means that A* algorithm used Euclidean Distance as the heuristic function. A* Euclidean means that A Star algorithm used Manhattan Distance as the heuristic function.

In the table, we could see that when size becomes large, the average time of returning an answer (whether solvable or unsolvable) increases. BFS2 and A* Euclidean are often the most time-consuming algorithms. We need to run the algorithm repeatedly for the purpose of validations. In order to make it faster in the following test, we chose 200 as the default size of our maze.

2 Find a random map with $p \approx 0.2$ that has a path from corner to corner. Show the paths returned for each algorithm. (Showing maps as ASCII printouts with paths indicated is sufficient; however 20 bonus points are available for coding good visualizations.)

TODO

3 For a fixed value of dim as determined in Question (1), for each $p = 0.1, 0.2, 0.3, ..., 0.9$, generate a number of maps and try to find a path from start to goal - estimate the probability that a random map has a complete path from start to goal, for each value of p. Plot your data. Note that for p close to 0, the map is nearly empty and the path is clear; for p close to 1, the map is mostly filled and there is no clear path. There is some threshhold value p_0 so that for $p < p_0$, there is usually a clear path, and $p > p_0$ there is no path. Estimate p_0 . Which path finding algorithm is most useful here, and why?

In this question, we tried to use a grid-search method first to narrow down the scope of parameters. We used 200 as the mazeSize, and we generated 800 different mazes for the validation of the parameter value.

After getting the result, we plotted the rate of successfully finding a path vs the p-value. See Figure 1.

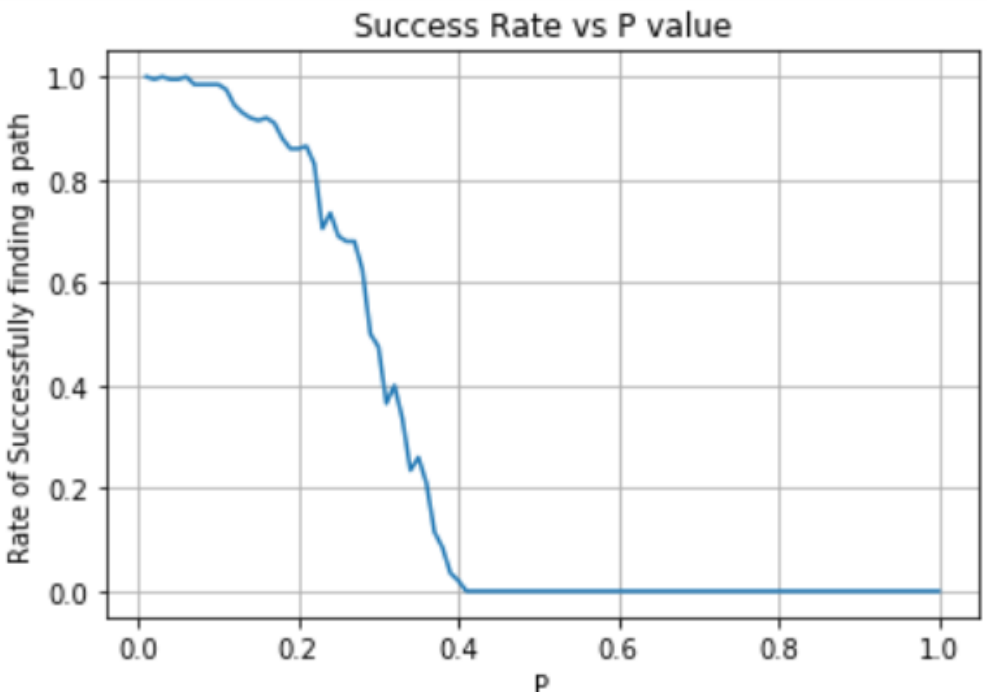

Figure 1: rate of successfully finding a path vs the p-value.

Then we tried to decrease the step size, and selected the p-value from 0.16 to 0.40 as the x-label to generate a new plot for better analyzing the threshold value here. See Figure 2.

After analyzing the result, we notice that the when p is at 0.40, the success rate stays at 0. Thus we came to conclusion that the threshold is 0.40 for p here. And to gain a 50% of success rate, the p-value should be set to 0.30.

In this problem, we used the Bidirectional A^* method, and the optimal distance function is Manhattan Distance. After the calculation, we found that BDA^* consumes less time than other algorithms. In this specific problem, because we simplified the settings, we could only move to 4 directions. Manhattan Distance better simulates such scenarios thus it has the best performance here.

4 For a range of p values (up to p_0), generate a number of maps and estimate the average or expected length of the shortest path from start to goal. You may discard all maps where no path exists. Plot your data. What path finding algorithm is most useful here?

5 For a range of p values (up to p_0), estimate the average length of the path generated by A^* from start to goal (for either heuristic). Similarly, for the same p values, estimate the average length of the path generated by DFS from start to goal. How do they compare? Plot your data.

6 For a range of p values (up to p_0), estimate the average number of nodes expanded in total for a random map, for A^* using the Euclidean Distance as the heuristic, and using the Manhattan Distance as the heuristic. Plot your data. Which heuristic typically expands fewer nodes? Why? What about for p values above p_0?

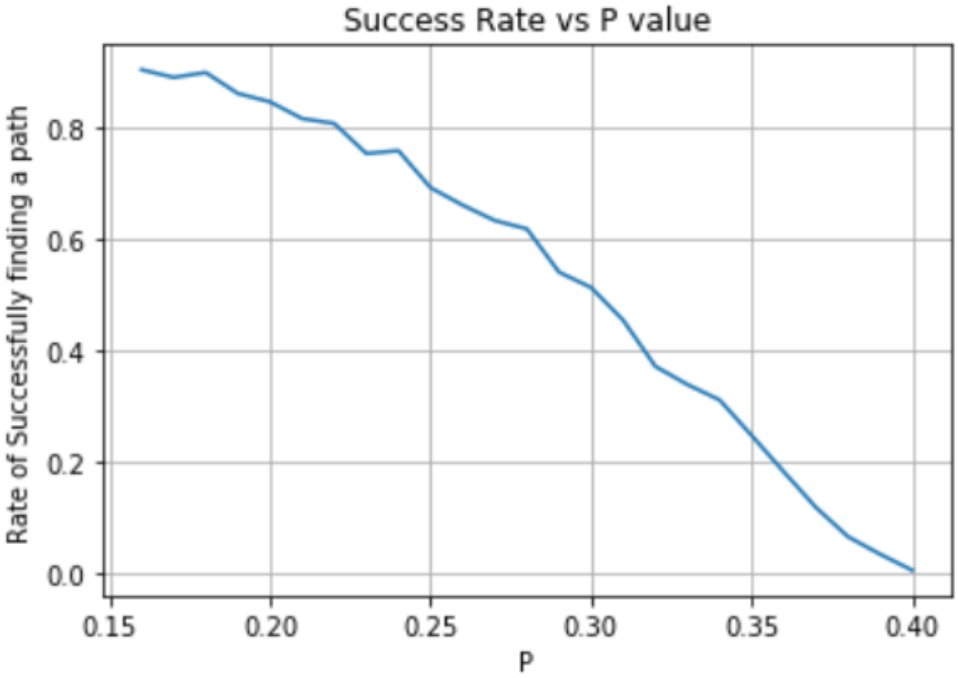

Figure 2: rate of successfully finding a path vs the p-value.

7 For a range of p values (up to p_0), estimate the average number of nodes expanded in total for a random map by DFS and by BFS. Plot your data. Which algorithm typically expands fewer nodes? Why? How does either algorithm compare with A^* in Question (6)?

Bonus 1 Why were you not asked to implement UFCS?

3 Part 2: Building Hard Mazes

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

2. Unlike the problem of solving a maze, for which the 'goal' is well-defined, it is difficult to know when we have constructed the 'hardest' maze. That being so, what kind of termination conditions can you apply to your search algorithm to generate 'hard' if not the 'hardest' mazes? What kind of shortcomings or advantages do you anticipate your approach having?

3. For each of the following algorithms, do the following: Using your local search algorithm, for each of the following properties indicated, generate and present three mazes that attempt to maximize the indicated property. Do you see any patterns or trends? How can you account for them? What can you hypothesize about the 'hardest' maze, and how close do you think you got to it?

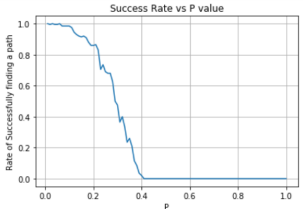

Figure 3: This frog was uploaded to writeLaTeX via the project menu.

4 Some LaTeX tips

4.1 How to Include Figures

First you have to upload the image file (JPEG, PNG or PDF) from your computer to writeLaTeX using the upload link the project menu. Then use the includegraphics command to include it in your document. Use the figure environment and the caption command to add a number and a caption to your figure. See the code for Figure 3 in this section for an example.

References

[1] K. Grove-Rasmussen og Jesper Nygård, *Kvantefænomener i Nanosystemer*. Niels Bohr Institute & Nano-Science Center, Københavns Universitet