Introduction to Artificial Intelligence

Shachar Zeplovitch, Syed Mahmood

February 2017

0.1 Introduction

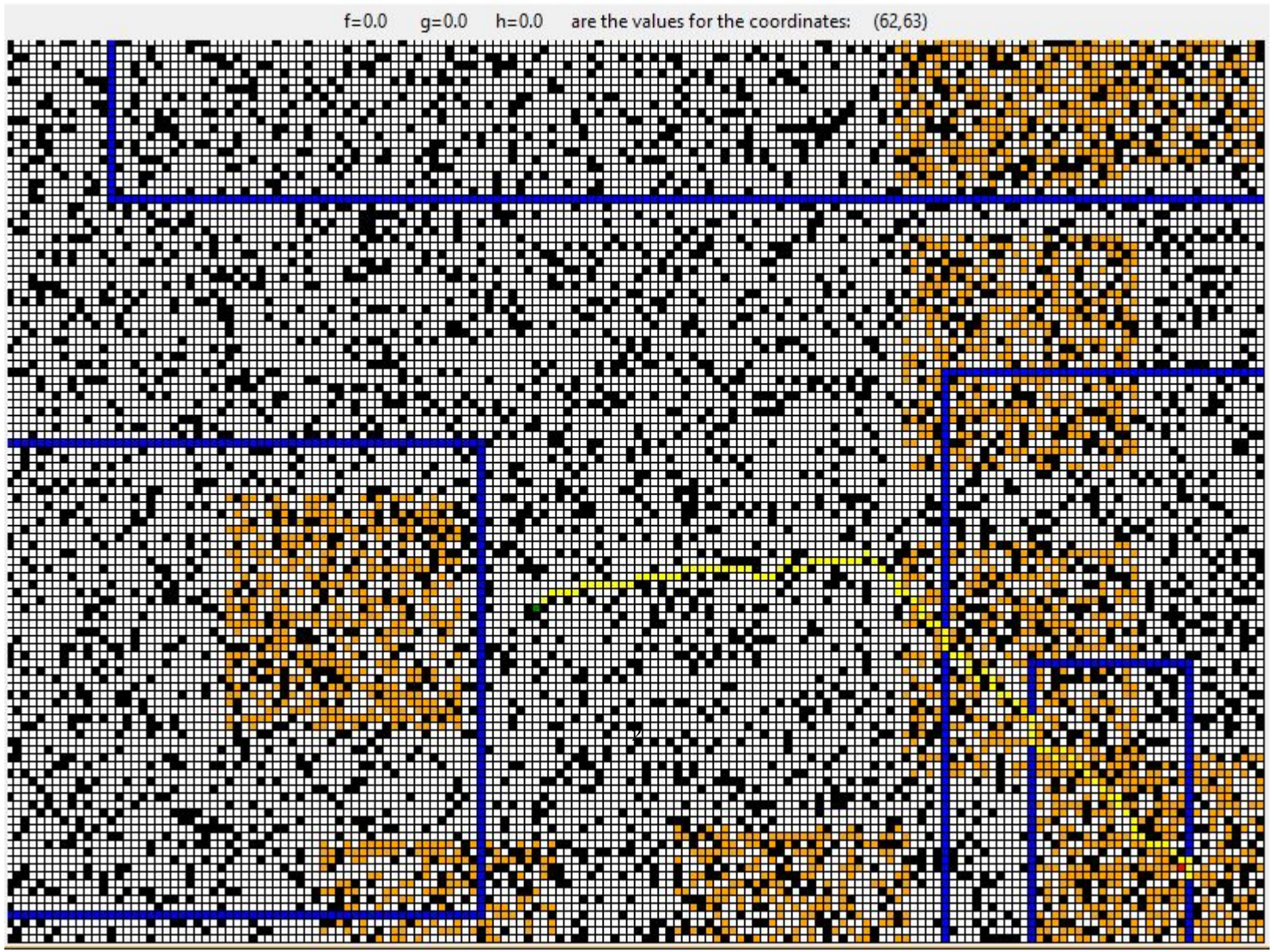
Our task in this assignment was to create a grid like world, with cells that aren't blocked, partially blocked, completely blocked, or serve as "highways" or rivers. Using A* and other algorithms, we are to compute a path from the start cell to the goal cell.

0.2 UI

A.

We made a grid as described in the assignment description. In order to make this grid we used the TKinter library, a UI library in the Python language. We display each of the cells with a different color corresponding to its type. The colors are as follows:

- White corresponds to unblocked cells
- Black corresponds to blocked cells
- Blue corresponds to highways
- Yellow corresponds to the path
- Orange corresponds to the hard-to-traverse cells
- Green corresponds to the start cell
- Red corresponds to the goal cell


f=0.0 g=0.0 h=0.0 are the values for the coordinates: (62,63)

0.3 Running the Code

There are four different files.

The files are `map_statifier.py, |map_generator.py,|`main.py, and searchalgorithms.py.

To run the code, make sure you create a directory "maps" in the same project and make sure all of the files are in the same project. Once you have the project set up and everything is at the correct place, run the following command(assuming Python 3 is installed):

python3 `map_statifer.py {ARG1} {ARG2}`

B.

Argument 1:

- 1 for Uniform Cost Search
- 2 for A*
- 3 for Weighted A*

Argument 2:

- 1 for Manhattan Distance
- 2 for Euclidean Distance
- 3 for Chebyshev Distance
- 4 for Diagonal Distance

0.4 Results

The following are the different types of heuristics we used:

(Manhatten Distance)

$$|x_1 - x_2| + |y_1 - y_2| \tag{1}$$

(Eucilidian Distance)

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{2}$$

(Chebyshev Distance)

$$\max\left(|x_1 - x_2|, |y_1 - y_2|\right) \tag{3}$$

(Diagonal Distance)

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{4}$$

(Diagonal Distance divided by 4 - Admissible/Consistent Heuristic)

$$(\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2})/4 \tag{5}$$

where (x1, y1) are the coordinates of the goal node. In order to analyze how well this algorithm worked, we see the time it took to compute a path from the start node to the end node. We also computed the path cost for each algorithm, and each possible heuristic.

Aside from the data that we collected, we also looked at the number of nodes that were expanded. On average, the number of nodes that were hit (i.e the yellow nodes) was the same as the amount required to go from the start to the goal in the shortest amount of expansions. So the shallowest path was always chosen.

Table 1: Average Execution Time (Seconds)

Algorithm	Manhattan Distance	Euclidean Distance	Chebyshev Distance	Diagonal Distance	Diagonal Distance divided by 4
A*	0.303	0.294	0.225	0.29	0.45
Weighted A* (W = 1.5)	0.20	0.24	0.216	0.238	0.42
Weighted A* (W = 2)	0.201	0.226	0.225	0.226	0.332
Uniform Cost Search	154.6	154.6	154.6	154.6	154.6

Table 2: Average Nodes Expanded

Algorithm	Manhattan Distance	Euclidean Distance	Chebyshev Distance	Diagonal Distance	Diagonal Distance divided by 4
A*	108.14	106	107.85	106	147.7
Weighted A* (W = 1.5)	101.0	98.66	104.4	98.66	137.2
Weighted A* (W = 2)	100.7	97.68	104.38	133	
Uniform Cost Search	13532	13532	13532	13532	

Table 3: Average Path Cost

Algorithm	Manhattan Distance	Euclidean Distance	Chebyshev Distance	Diagonal Distance	Diagonal Distance divided by 4
A*	122.6	115.6	128.94	115.6	162.8
Weighted A* (W = 1.5)	131.58	115.97	131.18	115.97	155.76
Weighted A* (W = 2)	131.93	118.24	131.46	118.24	149.764
Uniform Cost Search	186.32	186.32	186.32	186.32	186.32

0.5 Question C

We used python with this project, and the python standard priority queue is slow due to lacking an efficient update function. For that reason, we implemented our own priority queue so the update method runs much faster.

0.6 Analysis

D, E, F)

Algorithms Comparison(Execution Time, Nodes Expanded):

- Uniform Cost Search: This search algorithm returns an optimal cost but it is not feasible because it takes way too much time for it to finish. In addition, the number of nodes that are being expanded with this algorithm is way higher than the number of nodes in the path from start to goal.

- A*: The execution time of A* is shorter than the execution time of uniform cost search and also the number of nodes expanded is much lower than uniform cost search.

- Weighted A*: This is the fastest algorithm because it puts the most amount of emphasis on the distance from the current node to the goal which means less nodes will be expanded but the cost of the path might be higher than A* and Uniform Cost Search

- With A*, Chebyshev Distance and Diagonal Distance took the shortest amount of time(0.551 and 0.526). Euclidean Distance heuristic function came at last because it is the most inadmissible function out of the four. For the average number of nodes expanded, Diagonal Distance approach had the lowest amount of nodes expanded and Manhattan distance had the highest number of nodes expanded on average. Chebyshev had the highest path cost average, and Diagonal Distance approach had the lowest path cost average for A*.

- With Weighted A* (W=1.5), Diagonal Distance and Chebyshev Distance were computed the fastest on average, Euclidean Distance approach was the slowest of the four. Euclidean Distance function had the least amount of nodes expanded and Chebyshev distance function had the highest amount of nodes expanded. Euclidean and Distance approaches had the lowest average path cost and Manhattan distance at the highest average path cost.

- With A*, the Manhattan distance heuristic took the longest, followed by the euclidean distance, and finally the chebyshev distance was the quickest. With weighted A*, Manhattan and chebyshev took similar times to compute but euclidean took slightly longer.

- We used 5 different heuristics in our project. The best admissible/consistent heuristic we chose is the Diagonal Distance divided by 4 because in order to be admissible/consistent the function can never overestimate the cost and with that being said, since on highways the g cost is lowered by a factor of 4, we just divided the heuristic function to match that when the agent goes on the highway.

- For path cost, in A*, the Manhattan distance heuristic took the shortest amount of time, followed by euclidean, and then chebyshev. In weighted A*, the euclidean distance wins, and both Manhattan and chebyshev cost a lot more.

- A* allowed us to get the shortest path, i.e the path with the total lowest cost. This ensured that we chose highways whenever possible, and tried not to go for partially blocked cells. The Chebyshev and Euclidean distance heuristics can be inadmissible. The reason for this is because they assume there are no highways. Therefore, it is possible to overestimate the cost. Consider using the x distance or y distance; if the start and goal nodes are also on the same coordinates, and there are highways, then it is possible that the "estimated" distance is greater than the actual cost of traveling along the path.

0.7 Phase 2

0.7.1 Sequential A*

Average Running Time: 0.733
Average Nodes Expanded: 94.54
Average Path Cost: 116.3344

0.7.2 Integrated A* search

For Integrated A* search, the average number of nodes expanded was 140, path cost was 123, and average time was 0.65 seconds. Overall, the results are similar to A* and weighted A*, both in terms of the average time as well the number of nodes expanded.

0.7.3 Efficiency

Sequential A* search is efficient because there is more information that being utilized. Since we are using four different heuristics, including the manhattan distance, chebyshev distance, diagonal distance, and euclidian distance, there is a good amount of information we have at each node. Doing four searches and combining these results allows us to get a more accurate measure of the distance from the goal, which leads a more optimal solution.

Integrated has less nodes and time than Sequential. This is because sequential does as many searches as we have heuristics. Integrated does two at most for each of the cells.

0.7.4 Proof for part j

To prove i, we consider the cases where a cell can be expanded: There are at most two such cases. There are two types of search in integrated: anchor and inadmissible. Therefore, a cell can be expanded in either one, or both of these. Therefore none of the states are expanded more than once.

To prove ii, consider that any state that is expanded in the anchor search will not be in the i-th fringe anymore, because it was removed. This state cannot be revisited because it is no longer in the fringe, therefore it cannot be reexpanded.

To prove iii. we consider whether a cell can be inserted into a fringe. It will be inserted into the i-th fringe only if it has lower g value than before. And the node will be hit again only if it is reinserted into the fringe.