” Probabilistic Hunting”

Name: Muhammad Rizwan Khan (netid: mrk150)

1 Setup and Representation

All data storage and representation are done using two dimensional numpy arrays, with their indexes used as cell positions and their value storage used to store various values in the two-dimensional arrays. The value stored include probabilistic scores from computation, terrain assigned probabilities , false negative rates, belief and state of faith in cells etc.

1.1 Implementation:

Starting the program, you are presented with a list to start the algorithms. The list contains the choice to start part 1.1, 1.2, and modified algorithms for 1.4. The program displays terrains in a new GUI window and the total computations and total steps at the end of computation is displayed on the terminal The starting dimension is set to 50 by 50 for 1.1 and 1.2 but 15 by 15 for modified algorithm which are options three, four and five.

2 Explanation

The total information used during any given time, for any cell whether currently processed cell or any cell that will be processed etc. is the Observation. When we compute new probabilistic scores (new probability) of current search cells or targets in newer search cells, we update the information for all cells for future use, future reference and further computation for future use of probability, this includes our failure of finding the required cells, and using and updating information for future cells.

2.1 Start

Code operates on the initial state of assigning equal probability on each cell of all arrays we are using. Here we initialize four different arrays to store information for future use. We initialize an array for our Belief state, for our false negative rates for each of the cell that will be used in computation process in determining the probability in the update step, probability of the terrain type information of each cell (that is also used in generating for the graphic aspect of the code) and target in cells. When starting – the first operation does not have any

observation, success or failures on any of the cell searches. We simply open the cell, and then proceed to the process of updating our information

2.2 Update

This phase is the computation phase where all information for probability of cells, failures and success are determined solely here. Following steps happen during this process.

Check cell – If cell is the target cell we are searching for, we can stop our update process and proceed to the Finish state. If cell is not the target cell, then this would constitute a failure – we use this information in trying to determine the new location of the cell that is the target cell. This process includes:

- Take the value of our current cell from the belief array (as mentioned above all these arrays are using the same cell position to compute different information in trying to infer new data for the location) and update it for the current cell being searched (in this case – you can identify it as failure cell).

- Reduce probability of current cell by utilizing the false negative rates from the cell terrains – in particular, use the one for current cell:- *[false negative rate * probability of cells*]. Once done, recompute the changed information for the rest of the cells, by normalizing all cells :- *[probability of cell / total of all cells for all cells – more detailed below]*

2.3 Finishing

This is the step where we confirm our probabilistic score whether our current cell is the target. Here we utilize the information be gathered, and if true then we can stop the process as we have found the information. If not, we can repeat the update process and keep trying.
When computation is done and this phase is reached (from the update step), we do not stop the computation when one of the cell produces probabilistic score, rather it will rely on false negative rates of the cells that were produced during the computation during the update phase. In choosing whether if the current cell is the target, we randomly select a number between 0 and 1, and check if the false negative rate computation is less than the number we picked. If true, that determines that target is bound to exists. In any case if the target in the cell does not exist then we can continue with our further probabilistic computations, but usually it is able to determine that the target cell has been found through this.

3 Part 1

3.1 Part 1.1

When we start, the target is likely to be anywhere in the cells, and for that we use

$$P(TargetinCells) = \frac{1}{A * A}$$

A is the Dimension of the board. Following the 50 by 50 board, we start with 1 / 2500 as P(Target In Cell). Following our points from Explanation above, we make use of Start and Update and we can categorize as following:

- If current processed cell is the probability target cell we are looking for, and false negative rate determines it as False (hinting as not a false negative) then Target has been found
- If Current cell is not the target cell, means false negative rate has persisted and results in a failure. We move to the Update step in order to compute new probability for the rest of cells.

The following are used in computations

P(Target in cell) = P(Target In Cell) * P(False negative rate of the current cell terrain)

Belief state of cell = Belief state of cell * False negative rate of the current cell

As per the description, Target in cell , we are updating the Belief value of the current processed cell, and the False negative rate is the value we got for the False negative of the current cell (False negative rates are 0.1 for flat, 0.3 for hilly, 0.7 for forested and 0.9 for maze of caves – As per description we have to note false negatives are , P (Target not found in cell | $Targetisincell$).

As the probabilistic chance (probability) of current processed cells has now reduced (to find the target cell), we now perform the following step of normalizing, as the chance of finding our target cells has now increased in other cells in contrast to the declining probability of current cell. Note that, the total sum of new computed probability is not equal to one that is why we perform the step to normalize it, to try to edge the value closer to 1, which we are able to do using

$$\frac{P(TargetinCelli)}{SumofP(TargetinCells)}$$

3.2 Part 1.2

For the target to be found in the current processed cell, it would depend on

- Target being found in the cell that is being searched
- Probabilistic result whether if it's a successful or another failure.

In this case, take each cell on the board and consider it a terrain cell with the values from above description, meaning any of those cells can be cave, forest, flat or hilly. Consider these cells a more of a Target state or some state of Faith state where we would be able to find the target. In my approach, I set the Target state as the
Target State = 1 – False negative rate of the current cell picked (which can be false negative cell of any randomly assigned terrain and value – in this case 0.1 for flat, 0.3 for hilly, 0.7 for forested and 0.9 for maze of caves).

Note: Per my observation, you can either calculate the Target state at the very start when you are setting up probabilistic values for everything (setting values for each cell) or compute it after every update of the belief state, as long as the value for the cells and their false negative rates assigned to them are unchanged.
The following steps are computed to compute newer probability.

P (Target found in Cell $|observation) = P(TargetinCell$— observation) * P (Target found $|targetinCell)$

P (Target found in Cell | observation) = (1 – False negative rate of cell terrain) * P (Target in Cell)

Belief of cell = Target State * Belief of Cell

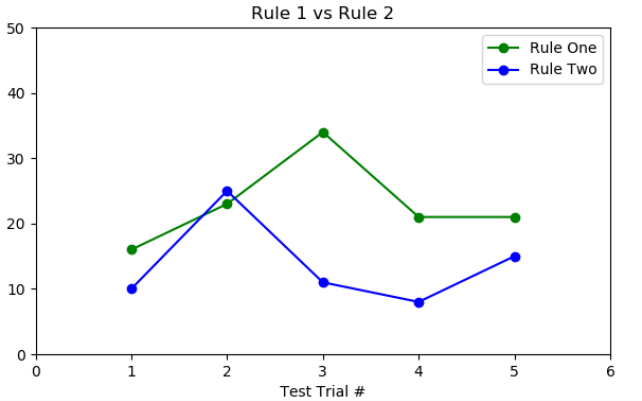
3.3 Part 1.3

Interpretation and Test Runs:
Rule 1: we pick cell with max belief value that has the highest probability of containing the target in its cell at given time t

Rule 2: Cell with highest probability of actually successfully finding the cell we are looking for - in simple terms, cell with the highest probability would be searched to find the target, at given time t (the cell we have the highest faith in).

In a random generated map of 25 by 25 and 50 by 50 Rule 2 seemed to perform somewhat better, where the iteration were sometimes better for Rule 1, but considering the randomity of board, overall Rule 2 performed better in a sense that it required fewer steps. For a more fixed and controlled generated board, we tested on a 5 by 5 dimension and while the comparison was close Rule 2 did have less steps taken to find the target than Rule 1. Graph below shows one of test performed on both Rule 1 and 2, and the average points out

that Rule 2 performed better due to less steps it took.
In the figure below, the blue line represents Rule 2 and green line represents line 1. Rule 2 is shown to be below Rule 1 indicating that it took fewer steps in comparison to find the target as compared to Rule 1. The result at point 2 is an indicator that despite carrying the results in a fixed map and random maps, the comparison came close on point 2 but overall Rule 2 did better. In the Figure below, the two points you see indicate the average steps of the two Rule. Note, the blue point is below green point as Rule 2 produced fewer steps than Rule 1 in the collected test data. In this, performance wise, due to lesser steps, Rule 2 performed better.



3.4 Part 1.4

As per requirement we always choose the cell with the minimal probability (Manhattan distance from current location)/(probability of finding target in that cell) – as per described in the project description. The steps for move is calculated it takes for the agent to go from current cell to the cell with the lowest probability score for cost distance computation.

3.5 Performance

Performance tends to be worse for the modified algorithm in comparison to the original rule 1 and rule 2, as we are now considering the distance cost and moving to cells to search and also considering the move action in the cost. This ends up giving worse performance than what Rule 1 and Rule 2 gives us. To give an example, multiple tests were carried out on 15 by 15 grid and Rule 1 produced a result where a target was found around 100 steps where as the modified one produced iterations of around 850.

But in comparison for modified Rule 1 and Rule 2, Rule 1 performed better in terms of steps it took to getting to its target in comparison to Rule 2. For Rule 3 where it directly searched the cell, it was close to Rule 1 but somewhat better.

3.6 Other:

One thing to note in this, is the issue of redundancy of the same probabilistic value appearing as the minimal score, which would cause us to go in an infinite loop, in order to solve it, as the probability of cells around would be also changed in belief state, and knowing that the redundant cell is a failure we chose to increase probabilistic score of that redundant cell slightly so it would give a chance to other cells and would not cause an infinite loop. Also, testing by both rule 1 and rule 2, setting the score slightly high, we could utilize in for false negative rates and our belief state which gave us the right answer. The downside of this approach is, our step cost is higher than rule 1 and 2 from 1.1 and 1.2 because we are considering the cost distance in the modified algorithm and every action of "moving" is consequential for the cost. In total, despite choosing the cell with the minimal cost of travelling, the cost function to determine cost values for cells still play part in this and the action to search or move simply increases the number of iterations/steps for us. Note: The probabilistic scores tend to spiral out of control when the board randomly generates large amount of hidden cave patterns (when 80 percent of map is filled with cave terrains) resulting in incorrect probabilistic score and causing loops. To deal with this issue, best approach for the third agent was to simply search the cell instead of relying on false negative or target cell (the state of confidence or hope that cell is a target cell) and deal with the failure if it is.

3.7 Own algorithm:

The only difference in my algorithm is it searches the cell, and the failure rate is kept in mind when computing the future cells, as it knows which cell not to search instead of relying on false negative rates like 1.4 for basic agent 1,2 and 3. This way I am able to bring down the steps it took to find the cell. On a few general tests carried out, on a 15 by 15 board, my algorithm managed to complete in 900 steps where as the other three were in the range of 1000 to 1200.

4 References

https://thispointer.com/numpy-amin-find-minimum-value-in-numpy-array-and-its-index/

https://www.tutorialspoint.com/python/tk$_b$*utton.htm*