Probabilistic Search And Destroy

Christopher Nguyen
Vatche Kafafian

April 10, 2021

1 Overview

In this project, we are going to use probabilities and utility functions to reinforce our beliefs to search for a target in a generated map with four different possible terrains for each cell.

2 Generating the Map

2.1 How The Grid Is Generated

The grid is generated as numpy matrix with values originally initialized to zero. Then, we randomly generate a value to initialize each cell to a terrain type. The different terrain types are flat, hilly, forested, and cave. Finally, we generate a random location for our target and generate the image for the map using matplotlib and marking the target with a red x. An example of the map is shown in the next subsection.

2.2 What The Grid Represents

After randomly generating the values for the grid we get this representation of a map shown below:

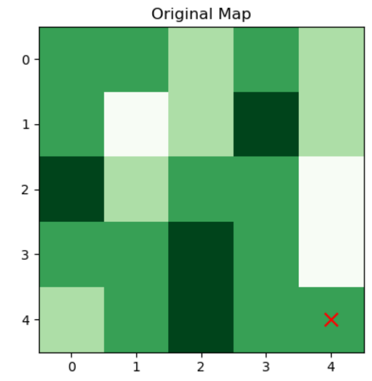

Figure 1: Original Map

The grid is represented with each cell containing a different shade of green. Each shade represents a different terrain, and as the shade gets darker the more difficult it is to find the target in that terrain. So the white cell would represent a flat terrain. The next lightest shade would represent a hilly terrain, and it would continue until reaching the darkest shade representing the cave terrain. In addition, the red x marked at the bottom right corner shows the targets location, in which this case shows that the target is in a forested terrain.

3 Problem 1

3.1 Belief State - Probability the target is in a given cell

For the first basic agent, we travel to the cell that is most likely to contain the target, using P(Target in Cell i—Observations t Failure in Cell j), and then we search that cell. The process is then repeated until the target is found in a cell. This process is split into three main parts, which consists of initializing, updating, and terminating the agent.

In the initialization state (t=0), the belief state is initialized, where the agent gives equal probability to all cells using the equation 1/ of cells. Then the agent checks the cell it is initially located at to see whether or not the target is in the cell. If it is not in the cell, we move on to the update state, where we calculate the probabilities of all the cells using the probabilistic formula above based off Bayes' Theorem.

In the update state, given all the observations at time t and the failure rate of a given cell, we determine the probability of a cell containing a target. First we check if the current cell contains the target if it does not contain the target, we update each cell in the belief state. We first lower the probability of the target existing in the current cell by using the false negative rate of the cell's terrain. Then by doing this the total probability is no longer equal to one so we normalize all the cells. For example, given a 4 by 4 grid all the cells initially in the belief state will have a probabilistic value of 1/16 for every cell. Then when we choose a cell and do not find the target we lower its probabilistic value, where P(Target in Cell | Observations t) is going to equal 1/16 for the prior cell or the cell we just searched. P(Failure in Cell j | Target in Cell i) is 0.3. So we lower the probabilistic value by doing (0.3 * 1/16). Now since the total probability is no longer 1, we normalize the belief state using the formula derived from Bayes' Theorem, where we get (0.3 * 1/16) / (15 * 1/16) + (0.3 * 1/16) = 0.0196 for the probability of the current cell and (1/16) / (15 * 1/16) + (0.3 * 1/16) = 0.0654 for the probability for the other 15 cells. This process is then repeated every time we determine that a cell does not contain the target. If we realize that the cell contains the target, we move on to the state where we find the target.

In the final state, we randomly generate a number between 0 and 1 to check if we will be able to find the target based on the false negativity rate of a terrain. If the the random number is greater than the false negativity rate, we determine the target to have been found and we conclude our search, if not we return to update and continue to search for our target until it is found.

4 Problem 2

4.1 stratTwoState - Probability the target will be found

For the second strategy, given all the observations from time t, we determine the probability of whether or not the target has been found in the cell when it is searched. We try to determine P(Target found in Cell i | Observations t). This is equal to P(Target in Cell i and Success in Cell i | Observations t). The success in cell i and the observations at time t are independent of each other. So we take Target in Cell i, which is what we know to be the belief state in the first strategy and we multiply it with P(Success in Cell i), which is equal to 1 - (False Negativity Rate of a given Terrain). By doing this after calculating the probability of the target in cell i, we are able to figure out the probability of a target being found in every cell. So Strategy two is added upon the foundation of strategy one, where we first calculate the belief state then use the formula (1 False Negativity Rate of a given terrain) Belief State [cell i] to calculate the new probabilites of each cell in the strategy two state and choose the highest probability. We repeat this until we move to our target state and find the target in order to stop our agent.

5 Agent 1 V.S. Agent 2

5.1 Agent 1

Agent one iteratively travels to the cell with the highest probability of containing the target, and searches that cell and repeats the process until target is found. This agent utilizes the formula in problem one to generate a belief state at each iteration t, which contains a probability for each cell containing the target based on passed observations t. We then represent this as well using matplotlib when printing our final results as shown below:

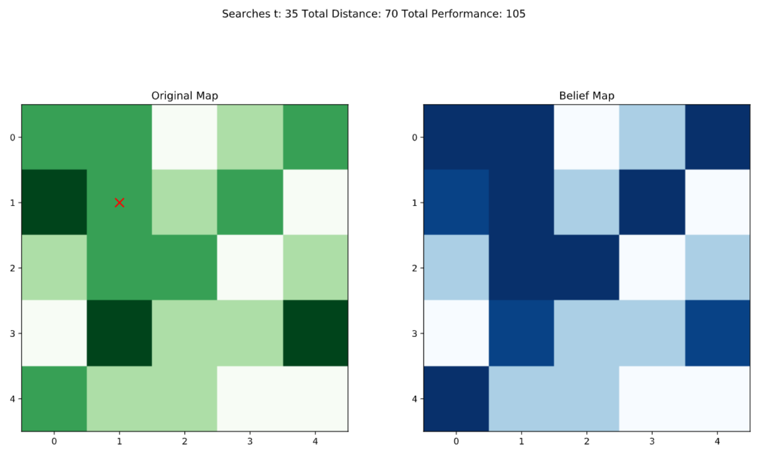

Figure 2: Belief State

As shown in the figure, the belief map is represented as a grid containing different shades of blue, where the darker shades possess a higher probability than the lighter shades just as it is in the original map.

5.2 Agent 2

Agent two iteratively travels to the cell with the highest probability of finding the target, and searches that cell and repeats the process until target is found. This agent utilizes the formula in problem two to generate a strategy two state at each iteration t, which contains a probability for finding the target in each cell based on passed observations t. We then represent this as well using matplotlib when printing our final results as shown below:

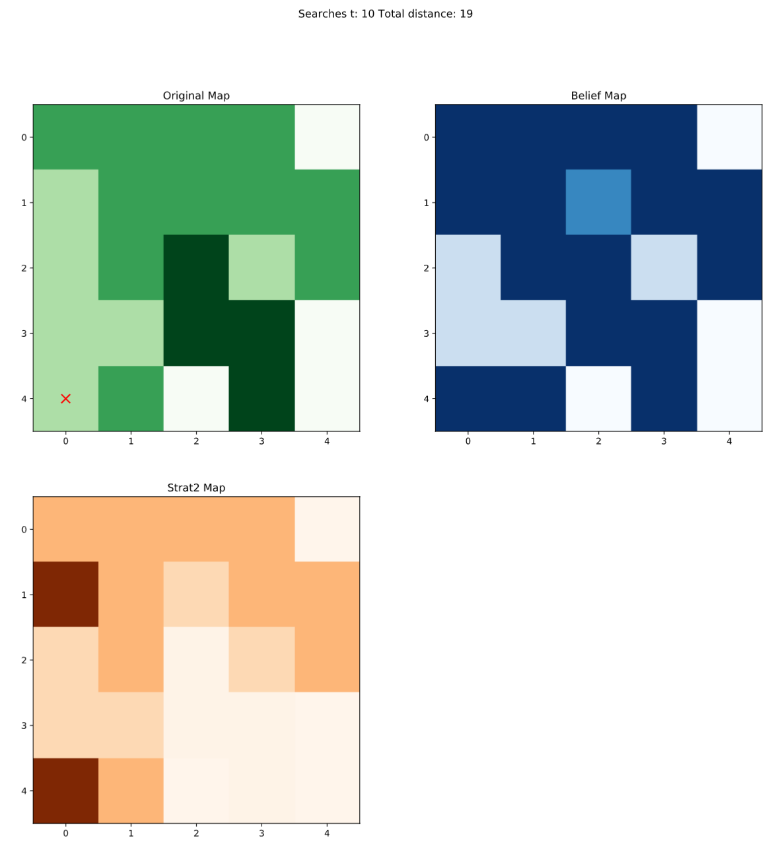

Figure 3: Strategy Two State

As shown in the figure, the strategy two state is represented as a grid containing different shades of orange, where the darker shades possess a higher probability than the lighter shades just as it is in the original map and belief map.

5.3 Belief State V.S. stratTwoState

In this subsection we are comparing both strategies and seeing which agent performs better on average. In order to calculate this, we randomly generated an original map ten times with a dim equal to five. In addition, the target and agent were randomly generated and set to a cell in the grid. After doing this and generating the map ten times for each agent, agent two ended up with a better total performance score. Agent one ended up with a search average of 92.2, a total distance traveled average of 249, and a total performance average of 341.2. While Agent two ended up with a search average of 53.3, a total distance traveled average of 172.9, and a total performance average of 226.2. Thus, agent two is better on average with a total performance average difference of 115. While calculating the averages for both agents, we realized that strategy two was more effective on average due to having a lower average on searches and distance due to finding the targets in terrains with a lower false negative rate. The time that both agents would perform similarly was when the target was located in a cave terrain. In the end, agent two performed better on average out of ten randomly generated maps.

6 Problem 4

6.1 Improved Agent

Our improved agent implements a few additions to agent one, which iteratively travels to the cell with the highest probability of containing the target within that cell and also takes into account the Manhattan distance from the current cell to all the other cells as well as the false negativity rate. We first find the the belief as we did with agent one and then for every cell in the belief state we divide the probability of cell i by the Manhattan distance plus the false negativity rate of every cell. After we finish, the improved agent picks the highest probability from the improved state to search next and we repeat until the target is found.
Unlike the first two strategies, our improved agent takes into account the distance traveled in sum with false negative terrain rate. We divide by this sum to account for the weight of the cost to search as well as find the new agent. The agent will be less likely to travel to farther distances, which will help reduce the distance cost. In addition, the false negativity rate is also added to the distance so that the agent will most likely travel to cells that are easier to search.

6.2 Improved Agent Representation

The improved agent is represented in the code using matplotlib as shown below:

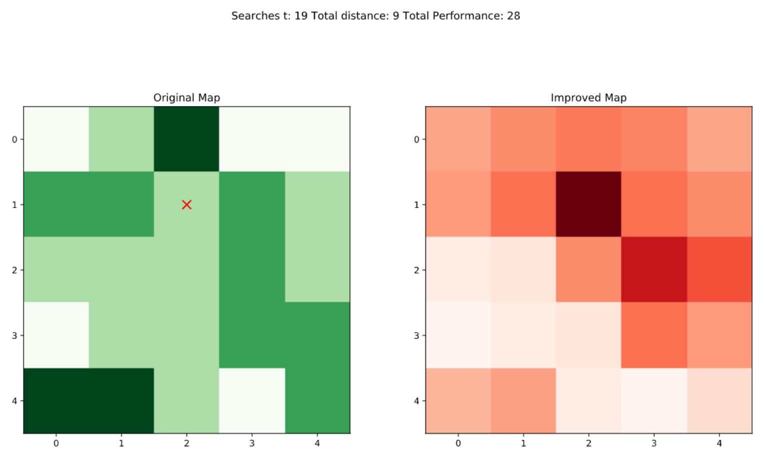

Figure 4: Improved State

The figure shows how the improved map is represented. Similar to the belief state and the second state, the improved state is portrayed using different shades of orange where the darker the shade the higher the probability. In the figure, we see how the darkest shade on the improved map represents the last highest probability chosen by the agent in which the target was found. In the next subsection we will see how our improved agent compares to the other two agents.

6.3 Improved Agent Results

In this subsection we are comparing the improved agent to both agent one and agent two, to see if the improved agent performs better on average. In order to calculate this, we randomly generated an original map ten times with a dim equal to five. In addition, the target and agent were randomly generated and set to a cell in the grid. After doing this and generating the map ten times for each agent, the improved ended up with the best total performance score on average. Agent one ended up with a search average of 84.1, a total distance traveled average of 231.2, and a total performance average of 341.4. Then Agent two ended up with a search average of 27.5, a total distance traveled average of 75.6, and a total performance average of 103.1. Finally, the improved agent ended up with a search average of 73.2, a total distance traveled average of 23.9, and a total performance average of 97.1. Thus, the improved agent is better on average with a total performance average of 97.1 compared to agent one with 341.4 and agent two with 103.1. While calculating the averages for agents, we realized that the improved agent was able to outperform the other two on average due to having a significant lower average on distance compared to the other two agents due to implementing the Manhattan distance in the utility function. By implementing the distance and weight of failure in finding the target, the improved agent was able to minimize the performance costs while searching for the highest probabilistic cell containing the target. Thus, the improved agent outperforms the other two agents on average due to its significantly lower distance traveled compared to the other two agents.

6.4 Improving Our Agent Based On Given Resources

One way that this algorithm can be improved given infinite time and space is to implement a Markov decision process. With this process, we would be able to map out the utility of moving from one cell to another, using a mathematical framework for modeling decision making in situations. One problem with this is that making one move to a certain cell may seem like a good decision, but it can affect the consequences of future decisions. For example if our Markov model had only one immediate action available to us, we moved potentially move to a cell with a high utility, but can consequentially leads us to moving farther from our target or to a path with bigger consequences in order to reach our target. In order to have a more accurate utility for each action, we would calculate the utility of an infinite amount of actions available to us including future actions given an infinite amount of time and space in order to find the most optimal path, minimizing the total performance costs, and finding the target. So by running many simulations of different paths using the Markov model, we can improve our agent by maximizing the possible efficiency and minimizing the cost given these resources.

7 Creative Acronym

The acronym we came up with for our improved agent is S.D.C.A.R.D. This stands for Search And Destroy Calculating Adjacent Rational Distance. We called it this because our agent searches for the target and destroys it by calculating our next cell based on distance choosing the most rational option adjacent to our current cell.