

CS 520 : PROBABILISTIC SEARCH AND DESTROY

Rithvik Reddy Ananth

11/18/2020

1 Problem Statement

A landscape represented by a map of cells. The cells can be of various terrain types ('flat', 'hilly', 'forested', 'a complex maze of caves and tunnels') representing how difficult they are to search. Hidden somewhere in this landscape is a Target.

The more difficult the terrain, the harder it can be to search - the more likely it is that even if the target is there, you may not find it. We are given the probabilities of target not found in a cell, given the target is in the cell as follows:

- 0.1 if the cell is flat
- 0.3 if the cell is hilly
- 0.7 if the cell is forested
- 0.9 if the cell is caves

Taking the false positive rate as 0, we have to find the target in the map.

2 Implementation

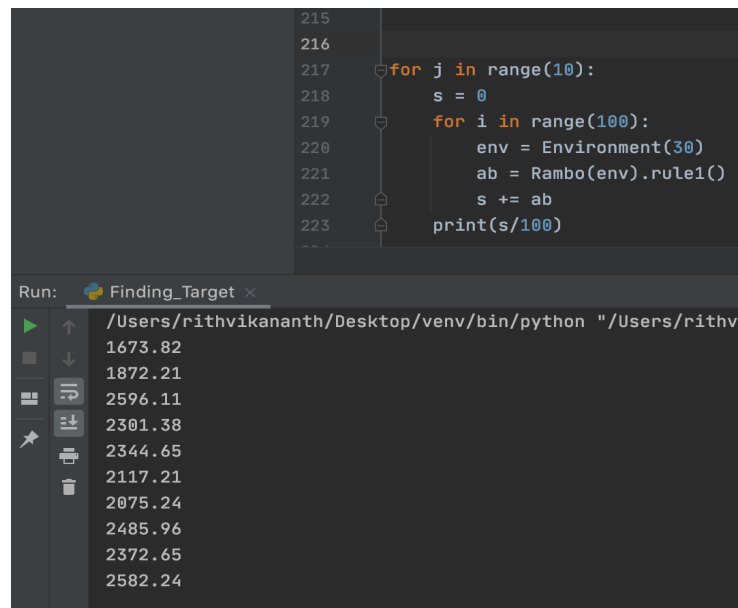
The map is a 30*30 matrix which are randomly assigned given terrain types (flat with probability 0.2, hilly with probability 0.3, forested with probability 0.3, and caves with probability 0.2).

A random cell is selected from the map and target is assigned to it. Each cell is a class having objects terrain, target (True only for the cell which has the target), belief, belief2, score and distance.

Belief is the probability of target in a given cell. Initially the belief for all the cells in the map is (1 / no of cells). **Belief2** is the probability of finding target in a given cell assuming that target is in the cell. Initialized the belief2 for each cell based on their terrain type (1 - false negative).

3 A Stationary Target

- Rule 1: At any time, search the cell with the highest probability of containing the target.



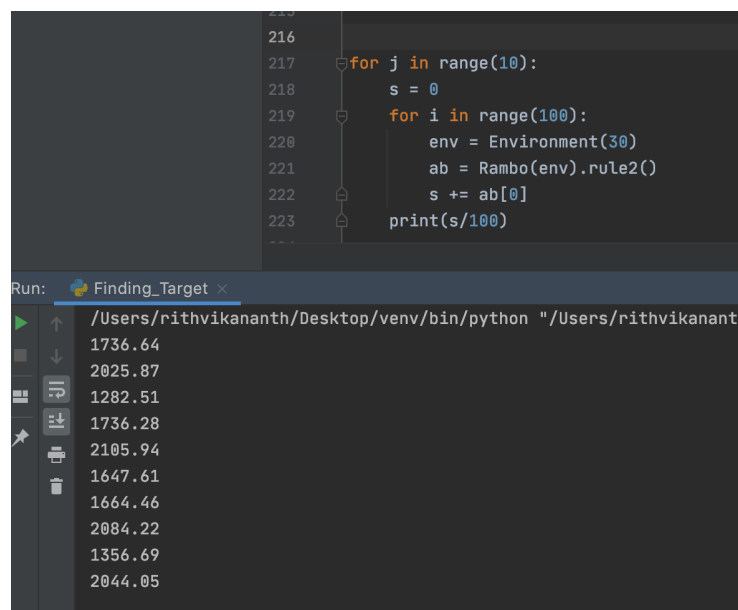
The screenshot shows a Python script in a code editor and its output in a terminal window. The script, located at lines 215-223, implements Rule 1: it iterates 10 times (j in range(10)), and for each iteration, it iterates 100 times (i in range(100)). In each inner iteration, it creates an environment (Environment(30)), runs the first rule (Rambo(env).rule1()), and adds the result to a sum (s += ab). Finally, it prints the average (s/100). The terminal output shows 10 average values: 1673.82, 1872.21, 2596.11, 2301.38, 2344.65, 2117.21, 2075.24, 2485.96, 2372.65, and 2582.24.

```
215
216
217 for j in range(10):
218     s = 0
219     for i in range(100):
220         env = Environment(30)
221         ab = Rambo(env).rule1()
222         s += ab
223     print(s/100)
```

Run: Finding_Target x
/Users/rithvikananth/Desktop/venv/bin/python "/Users/rithvikananth/Desktop/venv/bin/python"
1673.82
1872.21
2596.11
2301.38
2344.65
2117.21
2075.24
2485.96
2372.65
2582.24

Running the program using rule 1, we can see that on an average it takes 2241 searches to find the target in the map containing 900 cells. The above image shows the search averages to find the target for 100 iteration, and it is repeated for 10 times. We can see that the search averages are similar to each other without much variance.

- Rule 2: At any time, search the cell with the highest probability of finding the target.



The screenshot shows a Python script in a code editor and its output in a terminal window. The script, located at lines 216-223, implements Rule 2: it iterates 10 times (j in range(10)), and for each iteration, it iterates 100 times (i in range(100)). In each inner iteration, it creates an environment (Environment(30)), runs the second rule (Rambo(env).rule2()), and adds the result to a sum (s += ab[0]). Finally, it prints the average (s/100). The terminal output shows 10 average values: 1736.64, 2025.87, 1282.51, 1736.28, 2105.94, 1647.61, 1664.46, 2084.22, 1356.69, and 2044.05.

```
216
217 for j in range(10):
218     s = 0
219     for i in range(100):
220         env = Environment(30)
221         ab = Rambo(env).rule2()
222         s += ab[0]
223     print(s/100)
```

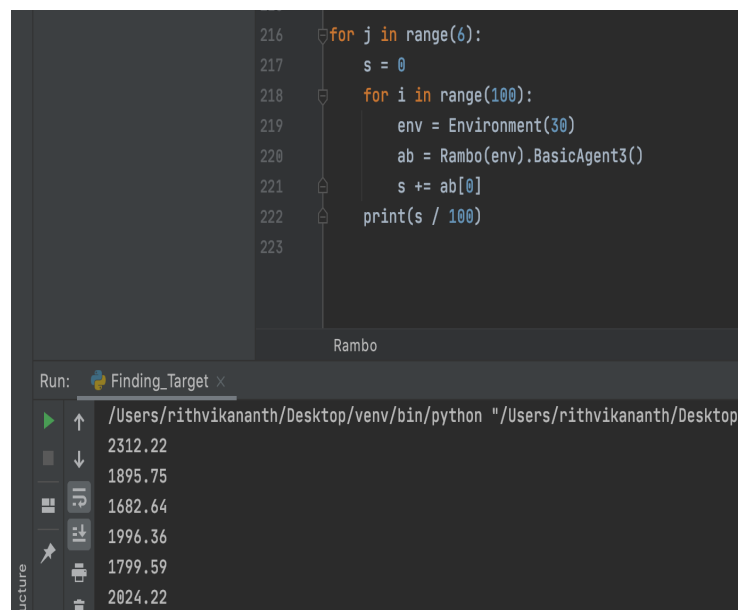
Run: Finding_Target x
/Users/rithvikananth/Desktop/venv/bin/python "/Users/rithvikananth/Desktop/venv/bin/python"
1736.64
2025.87
1282.51
1736.28
2105.94
1647.61
1664.46
2084.22
1356.69
2044.05

Repeating the same process as we've done for the above case, on an average it takes around 1767 searches to find the target cell using rule 2. Also search averages are similar to each other without significant variance.

Clearly searching a cell with highest probability of finding the target is ideal and that searching the cells with highest probability of containing a target takes 25 percent more searches and yes it holds across multiple maps.

- Basic Agent 1 travels to the nearest cell chosen by rule 1 and searches it. It takes 2479 searches for the basic agent 1 to find the target.
- Basic Agent 2 travels to the nearest cell chosen by rule 2 and searches it. It takes 1778 searches for the basic agent 2 to find the target.
- Basic Agent 3: We calculate the score for each cell, every time. The score is calculated in the following way:
$$\text{score} = (\text{Manhattan distance from current location}) / (\text{probability of finding target in that cell})$$

We identify a cell with minimal score and search it for the target.



The screenshot shows a code editor with a Python script and its output. The script is as follows:

```
216 for j in range(6):
217     s = 0
218     for i in range(100):
219         env = Environment(30)
220         ab = Rambo(env).BasicAgent3()
221         s += ab[0]
222     print(s / 100)
223
```

The output of the script, displayed in a terminal window, shows the average number of searches for 100 iterations, repeated 6 times. The values are:

```
2312.22
1895.75
1682.64
1996.36
1799.59
2024.22
```

Running the program using minimal score value, we can see that on an average it takes 1951 searches to find the target in the map containing 900 cells. The above image shows the search averages to find the target for 100 iteration, and it is repeated for 6 times. We can see that the search averages are similar to each other without much variance.

- Improved Agent: We can clearly see that the searches count is lower when we use rule 2. For basic agent 3 the score is calculated giving equal weight-age to the distance travelled by the agent and probability of finding the target in the cell. Now the score is calculated by giving more weight-age to the probability of finding the target in the cell assuming that travelling is not an issue for the agent. When we give 70 percent weight-age to the probability of finding the target and 30 percent to the distance to travel from current cell, the average number of cells to search are 1682. (I came to this conclusion after giving various weight-ages to the distance to travel and probabilities. One interesting find is that when more weight-age is given to reduce the travel distance from current cell, the average number of searches increases from basic agent 3. Also that there is no improvement in average searches if more weight-age is given to the probability of finding the target, so 70-30 ratio is ideal.)

4 References

- Lecture Videos
- Canvas Project discussion board
- Stackoverflow for solving errors in the code