CS 520: Introduction to Artificial Intelligence

Professor Wes Cowan

Assignment 3: Search and Destroy

Group Members:
Keya Desai (kd706)
Prakruti Joshi (phj15)
Rushabh Bid (rhb86)

1. Introduction to the problem

We have a landscape represented by a map of cells. The cells can be of four terrain types -

1. Flat
2. Hilly
3. Forested
4. A complex maze of caves.

There is a target hidden in this landscape and our goal is to find the target and destroy it. However, the difficulty of searching each terrain is different - the more likely it is that even if the target is there, you may not find it (a false negative). So we require a belief-based approach to determine where the target is more likely to be. The goal is to use the belief state to efficiently direct future actions.

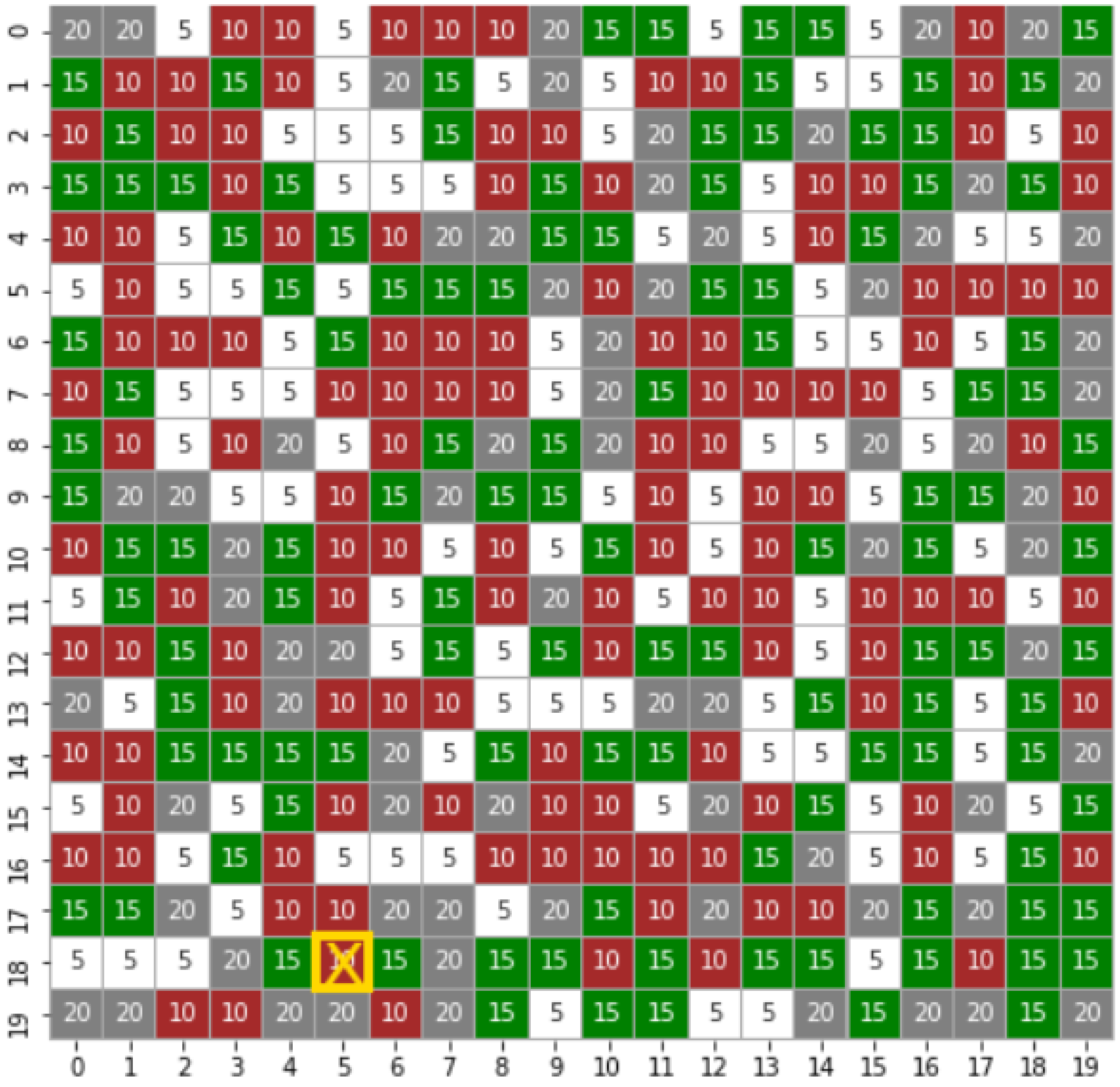
1.1 Representation

The landscape is represented as a numpy array of integer values. This map is called the environment grid *'env_grid'*. Each cell has a multiple of 5 as it's value. The values are as follows:

Cell value	**Terrain**
5	Flat
10	Hilly
15	Forested
20	A complex maze of caves

Table 1: Cell values and the corresponding nature of the cell

The location of the target is saved as a tuple of its coordinates -*'target_location'*. The environment is the combination of the environment grid and the target location and is stored as a tuple *'environment'* containing *'env_grid'* and *'target_location'*.



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	20	20	5	10	10	5	10	10	10	20	15	15	5	15	15	5	20	10	20	15
1	15	10	10	15	10	5	20	15	5	20	5	10	10	15	5	5	15	10	15	20
2	10	15	10	10	5	5	5	15	10	10	5	20	15	15	20	15	15	10	5	10
3	15	15	15	10	15	5	5	5	10	15	10	20	15	5	10	10	15	20	15	10
4	10	10	5	15	10	15	10	20	20	15	15	5	20	5	10	15	20	5	5	20
5	5	10	5	5	15	5	15	15	15	20	10	20	15	15	5	20	10	10	10	10
6	15	10	10	10	5	15	10	10	10	5	20	10	10	15	5	5	10	5	15	20
7	10	15	5	5	5	10	10	10	10	5	20	15	10	10	10	10	5	15	15	20
8	15	10	5	10	20	5	10	15	20	15	20	10	10	5	5	20	5	20	10	15
9	15	20	20	5	5	10	15	20	15	15	5	10	5	10	10	5	15	15	20	10
10	10	15	15	20	15	10	10	5	10	5	15	10	5	10	15	20	15	5	20	15
11	5	15	10	20	15	10	5	15	10	20	10	5	10	10	5	10	10	10	5	10
12	10	10	15	10	20	20	5	15	5	15	10	15	15	10	5	10	15	15	20	15
13	20	5	15	10	20	10	10	10	5	5	5	20	20	5	15	10	15	5	15	10
14	10	10	15	15	15	15	20	5	15	10	15	15	10	5	5	15	15	5	15	20
15	5	10	20	5	15	10	20	10	20	10	10	5	20	10	15	5	10	20	5	15
16	10	10	5	15	10	5	5	5	10	10	10	10	10	15	20	5	10	5	15	10
17	15	15	20	5	10	10	20	20	5	20	15	10	20	10	10	20	15	20	15	15
18	5	5	5	20	15	X	15	20	15	15	10	15	10	15	15	5	15	10	15	15
19	20	20	10	10	20	20	10	20	15	5	15	15	5	5	20	15	20	20	15	20

Figure 1: Environment Grid Representation with target at (18, 5)

The agent uses another matrix called the *'belief_map'* that contains the measure of our belief of each cell having the target. This matrix is actually a numpy array of float values of belief. We keep updating this matrix based on the information we get after each search. The selection of the cell to be searched at every step is decided based on a particular rule- Rule 1 or Rule 2 which are described below. The total number of searches the agent takes to find the target is tracked in a counter*'num_searches'*.

2. Belief Update

Initially, the target is equally likely to be anywhere in the landscape, hence starting out, the prior belief of each cell having the target is:

$$P(Target\ in\ cell\ i) = \frac{1}{No.\ of\ cells\ in\ the\ map}$$

At any given point we search the cell that has the maximum likelihood for reward. Since, in the beginning, all beliefs are equal, it does not matter which cell we explore. Hereafter, observing the result of each consequent search, we can update our beliefs accordingly. The problem that now poses is how to factor the observations to adjust our beliefs appropriately. The Bayes' theorem is applied for updating our belief state.

Given observations till time 't+1' i.e. observations till time 't' and failure on searching a cell 'j'

($Observations_{t+1} = Observations_t \wedge Failure\ in\ Cell\ j$), the belief of target at any cell 'i' is given by:

$$P(Target\ at\ i \mid Observations_t \wedge Failure\ in\ j)$$

$$= \frac{Prior\ belief\ for\ cell\ i * Y i}{Prior\ belief\ for\ cell\ i \times False\ Negative\ rate\ of\ j) + (1 - Prior\ belief\ for\ cell\ j)}$$

where,

$Y_i = 1,\ if\ i \neq j$

$Y_i = False\ negative\ rate\ of\ terrain\ at\ i,\ if\ i = j$

At this point, we have successfully accounted for the observations so far while computing our updated belief state. The current belief state depicts the likelihood of the presence of the target. However, the presence of the target is not the only primary factor we need to consider. Since searching each terrain is not equally easy or difficult, we also need to factor in the success rate of finding the target in a cell given the target is present there. In this case, instead of the probability that the target is at a given cell, we use the probability that the target will be found at a given cell. Computing these probabilities will require the false negative rates for the different terrains.

$$P(Target\,found\,at\,i \mid Observations_t) = P(Target\,at\,i \mid Observations_t) * Y_i$$

where,

$$Y_i = False\ negative\ rate\ of\ terrain\ at\ i$$

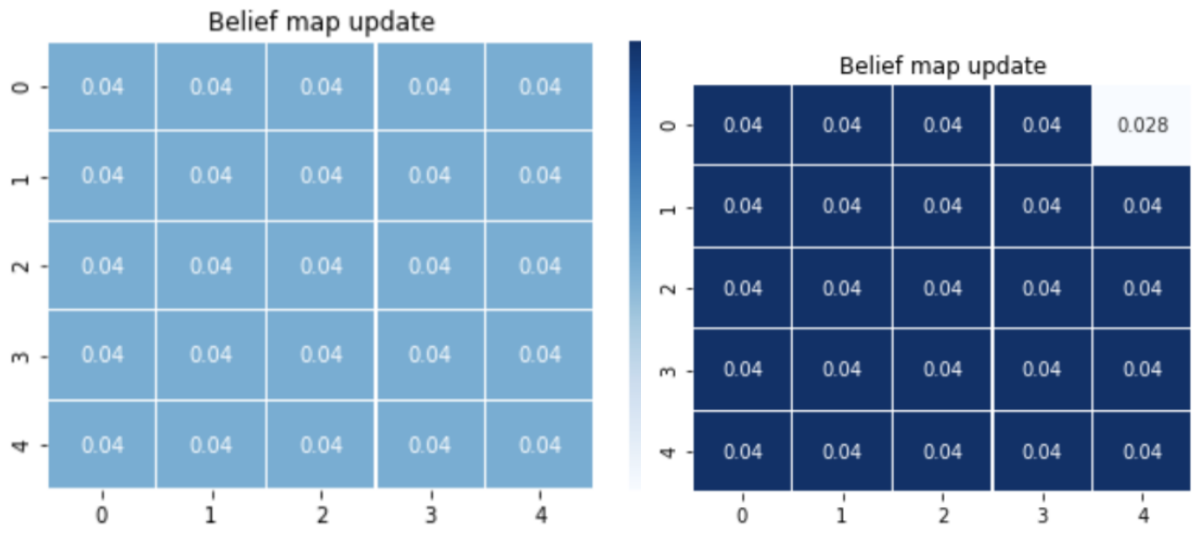


Figure: Belief update for a (5 x 5) environment after failure in search at cell (0,4) of Terrain Type Forested.

3. Decision Rules

Consider comparing the following two decision rules:
{ Rule 1: At any time, search the cell with the highest probability of containing the target.
{ Rule 2: At any time, search the cell with the highest probability of finding the target.
For either rule, in the case of ties between cells, consider breaking ties arbitrarily. How can these rules be interpreted/implemented in terms of the known probabilities and belief states?
For a fixed map, consider repeatedly using each rule to locate the target (replacing the target at a new, uniformly chosen location each time it is discovered). On average, which performs better (i.e., requires fewer searches), Rule 1 or Rule 2? Why do you think that is? Does that hold across multiple maps?

When observed carefully, the first rule simply states that select the Cell *'i'* that gives the maximum value for $P(Target\ at\ i \mid Observations_t \wedge Failure\ in\ j)$ while the second rule prefers the Cell *'i'* that maximizes $P(Target\ found\ at\ i \mid Observations_t \wedge Failure\ in\ j)$. Thinking about the two rules logically, it can be noticed that the first rule only considers the belief about the target being in a certain cell while the second rule factors in the difficulty of each terrain. Since the second rule uses more information for inference, it could be one reason to believe that it would perform better. However, it is better to analyze the risks rather than making decisions based on assumptions. Table 2 shows the average number of searches required to find a target using rule 1 and rule 2. It can be observed that on an average, Rule 2 seems to perform better as it takes less searches to find the target.

	Rule 1	**Rule 2**
Average number of searches on the same map with changing target location	6399.86	5586.44
Average number of searches on multiple maps	6099.45	5133.66

Table 2: Comparison of Rule 1 and Rule 2 in terms of average searches required
(Average over 50 simulations on maps of size [50 * 50])

The reason for the second rule performing better than the first one might seem counter-intuitive. The first rule prefers the cells that have the higher chance of containing the target, while the second one prefers the cell that give higher chances of finding the target. Now, considering the higher chance of finding the target might seem like a good idea because it makes sense to explore a place that is easier to explore. However, we succeed only when the target is present there. Thus, finding a target in a cell which is easier to find but with quite low probability of target actually being present seems futile.

The arguments above imply that Rule 1 should work better than Rule 2. Let's consider a grid of size 10×10. Initially, the belief for all cells would be 10^{-2}. Now, the cells with flat terrain give a better probability of finding the target. The second rule advises that the flat cells should be searched first. The probability of a cell being flat is 0.2. Since the target is arbitrarily placed in any cell, in 20% of the maps the target will be in a flat cell, and therefore, Rule 2 will work better for these maps because Rule 1 will search the other three terrains as well. Moreover, the probability of a cell being hilly is 0.3 and the false negative rate is also 0.3. So, the hilly cells also give a good chance of finding the target given the target is in them. So the second rule would work better for hilly terrain as well because it would search the hilly cells when it is done with the flat ones. Rule 1 on the other hand would still take more searches as it looks at the other two terrains. In total, the flat and hilly cells add up to one half of the landscape, so it is effectively inferred that Rule 2 works better for at least half the maps. Figure 2 shows the plot of average searches required for Rule 1 and Rule 2 on the same map as well as different map. It can be observed that Rule 2 performs better in both cases. Thus, our reasoning is backed by statistics obtained and it stands for multiple maps too. Thus, we can generally state that Rule 2 would perform better than Rule 1. Further arguments based on the difficulty of the terrains are made in section 5.

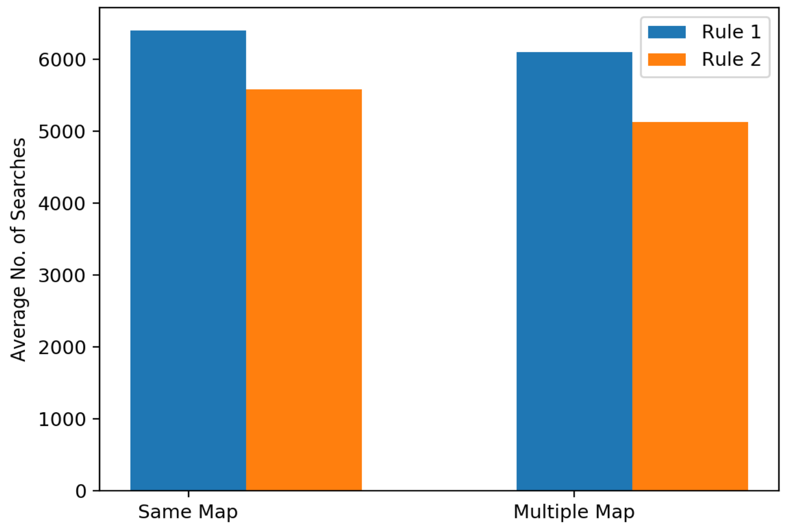


Figure 2: Bar-plot for average searches on same and multiple maps

4. Minimizing Actions

Consider modifying the problem in the following way: at any time, you may only search the cell at your current location, or move to a neighboring cell (up/down, left/right). Search or motion each constitute a single `action'. In this case, the 'best' cell to search by the previous rules may be out of reach and require travel. One possibility is to simply move to the cell indicated by the previous rules and search it, but this may incur a large cost in terms of required travel. How can you use the belief state and your current location to determine whether to search or move (and where to move), and minimize the total number of actions required? Derive a decision rule based on the current belief state and current location, and compare its performance to the rule of simply always traveling to the next cell indicated by Rule 1 or Rule 2. How does Utility apply here? Discuss.

4.1 Defining the cost function

Till now, our agent has been teleporting to the best cell to search. But now travel comes with a cost. Every search and move action incurs a cost of one and the aim is to minimize the number of actions required in searching the target. If we simply travel to the next optimal cell, indicated by previous rules, it might incur a large cost since earlier it did not take into account the distance needed to travel. It might be the case that there are cells near the current cell, in which the agent has less belief but incurs less cost as it is nearer. For the same cost of traveling to the further cell with more belief, the agent might be able to search in multiple cells near to its current position.

To measure the number of actions required to travel and search a cell i and the reward of finding the target there, we define a cost function, which takes into account:

1. manhattan distance of cell i from the current position of the agent,
2. agent's belief of finding the target there - $Belief(i)$ and, the probability of finding the target in cell i on searching if the target is present in cell i (*1- False negative rate),*
3. the action of searching the cell i *(1)*

Suppose that the agent is currently at location *'current_cell'*, one way of formulating the cost function is as follows:

$$One\ Step\ Cost_{current\ cell}\ (cell\ i) \ = \ \frac{Distance(current\ cell, cell\ i)}{2 \times dim} + [\ 1 \ - \ (Belief(i) \ \times \ (\ 1 - \ Y_i))\ ] \ + 1$$

where,

$$Y_i \ = \ False\ negative\ rate\ of\ Terrain\ at\ \ i$$

Here, the distance from the current cell to another cell is normalized by the maximum manhattan distance possible i.e. two times the dimension of the map. This normalization helps to include the effect of both the distance and the probability of finding the target in the cell by scaling the values appropriately.

The agent can make a decision of which cell to search next based on the cost function defined above. The agent picks the cell with minimum estimated cost to search next. The agent explores this cell, updates its belief map if the search is a failure and recomputes the cost of moving from the new position.

One interesting case that might come in this scenario is this: Suppose that the agent is currently at cell m. The minimum cost computed using the above-mentioned method comes out to at cell j. Now, there might exist a cell i on the way to cell j which has only a slightly higher cost than cell j. If the agent does not find anything at cell j, it will return to cell i eventually. This will add on to the cost in the form of additional distance traveled by the agent. A better approach might be to search cell i on the way, and then move on to cell j if it still has the minimum cost.

The most basic requirements for this are the cell that we most recently explored and the one with the minimum cost at the given belief state. Let's call these cells the *'initial cell'* and the *'final cell'* respectively. Now, imagine a hypothetical rectangle with the *'initial cell'* and the *'final cell'* on diagonally opposite corners. Let's call this rectangle the *'area of free movement'*. Since, movement is only allowed horizontally and vertically, going through any one of the cells in the *'area of free movement*' does not add to the final distance between the *'initial cell'* and the *'final cell'*. So if we add one more search while travelling from the initial cell to the final cell, the only additional cost incurred is of the search. Let's cell the cell with the minimum cost within the *'area of free movement'*, the *'checkpoint'*. It is a safe bet to state that the *'checkpoint'* would be a valuable additional search on the way to the final cell. But, can we do better?

Consider drawing a horizontal and a vertical line passing through the *'checkpoint'*. This divides the *'area of free movement'* in four parts. We discard the parts that do not contain the *'initial cell'* or the *'final cell'*. This gives us two parts, one with the initial cell and the checkpoint on the opposite corners, and the other with the checkpoint and the final cell on the opposite corners. These two parts look like a smaller version of the original problem. Taking the minimum cost in each of these parts can find two more potential checkpoints, one in each part. Repeating the procedure, more and more checkpoints can be generated. However, there should be a way to control the number of checkpoints generated. Also, there is a need for validation of the checkpoints. A cell giving the minimum cost in a particular region does not necessarily mean it is worth exploring. Thus, this approach is usable when formulated correctly but if the analysis is not done carefully, the cost added in terms of redundant checkpoints becomes huge.

A more general and potentially better approach built on this intuition is to consider all the pairs of (cell *i*, cell *j*) values and then select the cell i which has the minimum expected future cost. The agent is effectively looking two steps ahead while making the decision for selecting the next cell for the current cell. This includes the notion of utility as the cell with minimum expected cost (considering the expected cost from future steps) will have the maximum utility. The value or the utility of moving to a particular cell is estimated by taking into consideration the minimum future cost.

The cost function of the two step look ahead approach can be defined as follows:

$$Two\ step\ Cost_{current\ cell}(cell\ i,\ cell\ \ j)$$

$$= \frac{Distance(current\ cell, cell\ i)}{2 \times dim} + 1 + [\ (\ 1\ -\ (Belief(i)\ \times\ (1\ -\ Y_i))\ )\ \times Cost(j)\ ]$$

$$= \frac{Distance(current\ cell, cell\ i)}{2 \times dim} +\ 1\ +\ [\ (\ 1\ -\ (Belief(i)\ \times (1\ -\ Y_i))\ )\ \times\ min_k\,(One\ Step\ Cost_j(k))\ ]$$

where,

$$Y_i\ =\ False\ Negative\ Rate\ of\ Terrain\ at\ i$$

Here, the cost pair *Cost(cell i, cell j)* is defined with respect to the current cell that the agent is in. The estimated cost function for all the cells looking two steps ahead is computed with reference cell as *i.* This factors in the expected value of the second cell (here *j*) which is again a one step estimation cost of the second cell.

The pair with the minimum value of cost function is selected. In case of pairs having the same value of cost function, ties are broken by selecting a random pair. The agent then moves on to the first cell in the pair, updates its belief map, re-computes the two step estimated cost from the new position to every other pair of cells and thus proceeds in a similar manner to decide the future actions.

4.2 Analysis

Comparing the performance of decision rule based on estimated one and two step costs to the rule of simply always traveling to the next cell indicated by Rule 1 or Rule 2:

The essence of decision rule derived in section 4.1 is to move to the cell with minimum estimated cost which takes into account the distance of a cell from the current position of agent. If we ignore the number of actions incurred by travelling to a cell, and move to a cell with maximum belief or confidence i.e based on rule 1 or rule 2, then the total number of actions ought to shoot up. The result in Figure 3. is in line with this intuition. Using the decision rule based on both one-step and two-step perform better (take less actions in searching the target) than using the decision rules of 1 and 2, across all terrain types. This establishes that the cost functions correctly evaluates the value of moving to a particular cell in the grid such that the expected future costs (in terms of actions) are less.

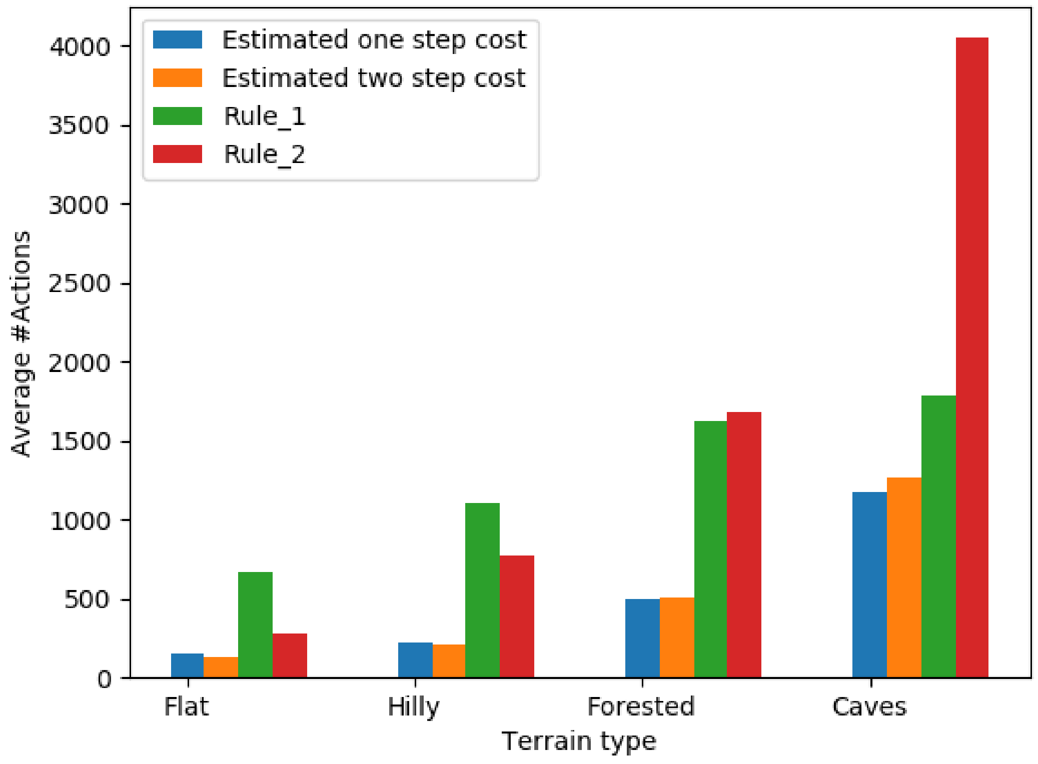


Figure 3. Average number of actions required to search for the target in a map of 10 * 10 using the two cost functions on the same map for each terrain type. (#simulations = 500)

5. Terrain wise Analysis

An old joke goes something like the following:
“A policeman sees a drunk man searching for something under a streetlight and asks what the drunk has lost. He says he lost his keys and they both look under the streetlight together. After a few minutes the policeman asks if he is sure he lost them here, and the drunk replies, no, and that he lost them in the park. The policeman asks why he is searching here, and the drunk replies, "the light is better here". In light of the results of this project, discuss.

The jokes take us back to the debate on the performance of the two rules. Let us first analyse the behavior of the drunk man. He knows that he is drunk and might have lost them anywhere but has a vague memory that he lost them in the park. However, since he is drunk, it is sensible for him to start at a place that is easy to explore (under the streetlight). Assuming the drunk man is sober enough to later realize that the keys might be at some other place, we can relate him to be following the second rule. Starting at a place that is easier to find and moving to places in increasing order of their exploration difficulty. However, since he had a vague idea that he lost them in the park, much like our case, the chances of finding the keys under the light are comparatively slim.

Generally speaking, it would be a good idea to start with the easier option. This holds because when you start looking at a place that is easy to explore, when you don’t find it there, you can be comfortably confident that the object is not present there. Thus, when you move on from one place to the other, you can assume that you would not have to return there. If the drunk man were to start looking in the park, since the lighting there would be low, there is a high chance he would not find it there. Now, he would randomly move to another place and start looking for it. In this case, when he searches at a place that is easy to explore, he can still be sufficiently confident that the object is not there. But since he does not have an order planned to search, by the time he looks at a few more places he would start doubting his previous explorations and consider going back there. This is not the only problem that can happen. The other major issue with this unplanned approach is that when you move on from a place thinking the target is not there, you stay away from it for a long time because you would rather look at all the other places that have not been recently explored. Thus, the approach where he searches the easy to explore places first, can prove helpful. Here the different places can be thought of as different terrains in our landscape. The false negatives can be a result of poor lighting in this case. If we compare the two searching rules discussed earlier for targets in specific terrains, we should be able to verify our assumptions and arguments.

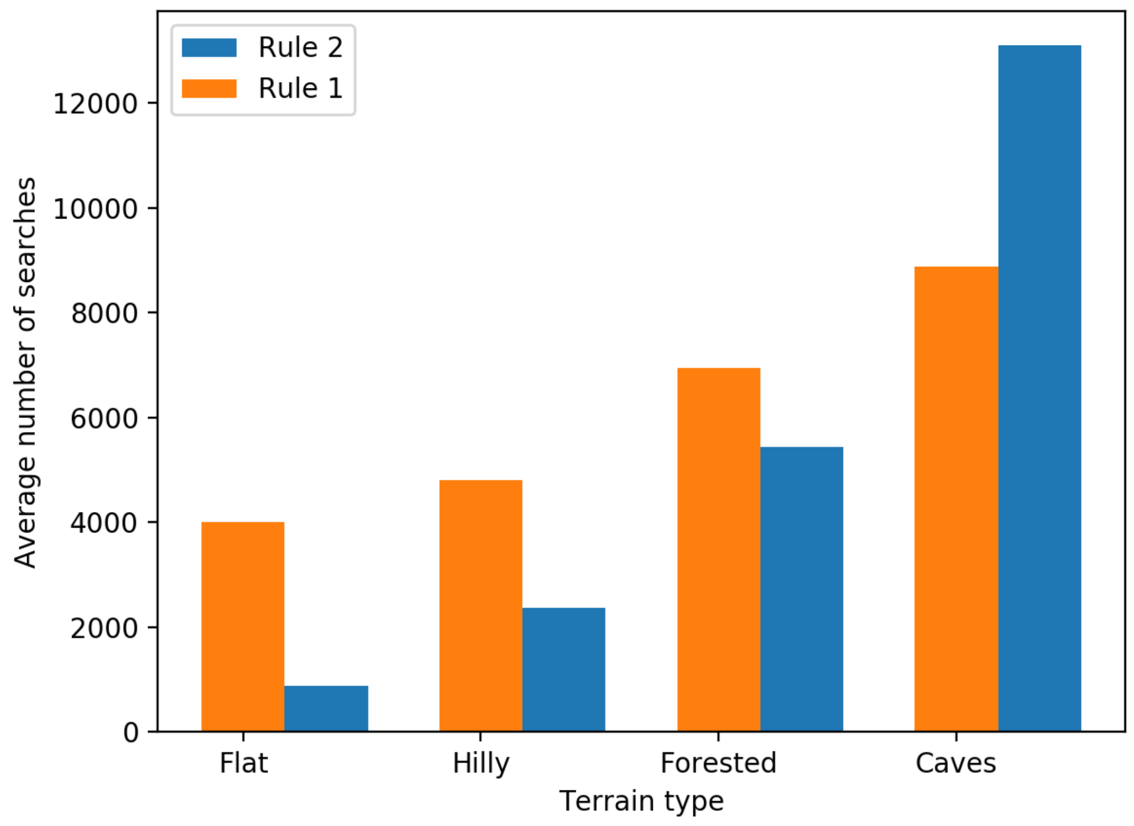


Figure 4: Terrain wise performance of the two rules (Bar-plot)

As we see in the plot above, the assumed behaviour is observed when the target is deliberately placed in a flat, hilly or forested terrain. The average number of searches taken by following the approach of Rule 1 and Rule 2 is observed in each terrain. In complex mazes of caves, the first rule seems to be running better. This seems plausible as when the target is in caves, and you start searching in increasing order of exploration difficulty, the second rule does substantial work on the terrains that do not have the target. The first rule on the other hand does have a small bias towards opening explored but difficult cells multiple times, however it has a tendency to jump from terrain to terrain somewhat arbitrarily. Even though the second rule fixates on an order and can give an approximate idea about when the target will be found, when the target is in the last place where we will look, the searches taken will be very high. The first rule on the other hand is more likely to find the target while arbitrarily jumping from terrain to terrain.

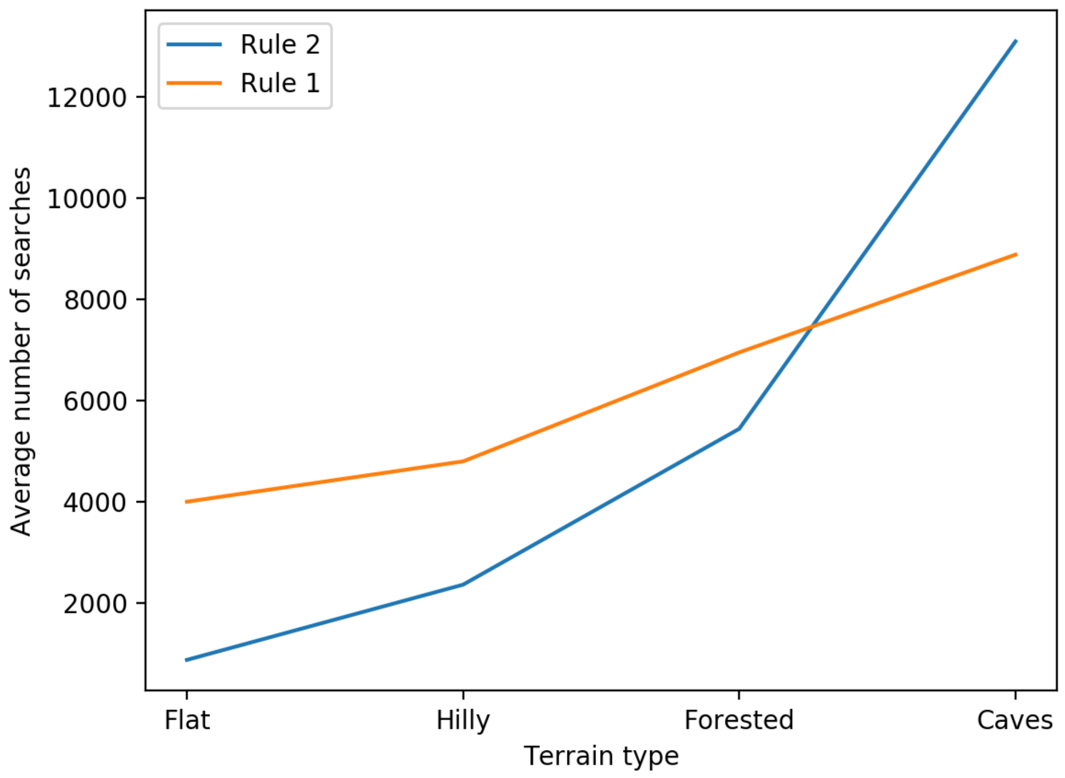


Figure 5: Terrain wise performance of the two rules (Line-plot)

The line plot above better depicts the number of searches given by the two rules as a function of terrain difficulty. When the terrains are easier to explore, the second rule gives more optimal results as compared to the first one. We can see the two lines intersect near the forested terrain. This means that the two rules become comparable near the forested terrain. We also see that the first rule overtakes the second rule as terrain difficulty further increases. We can conclude two things based on these analysis. Firstly, in general terms, it is safe to assume that the second rule would give a better result than the first one. Finally, the drunk man accidentally or cleverly, sticks to the correct approach to his problem and we can learn a lesson from him.

6. Bonus: A Moving Target

In this section, the target is no longer stationary, and can move between neighboring cells. Each time you perform a search, if you fail to find the target the target will move to a neighboring cell (with uniform probability for each). However! All is not lost; your target has a tracker that sends you information about their position. The tracker is meant to send back the type of terrain the target is located; however, it is broken, and each time step it reports a terrain type where the target is not. (i.e., if the tracker reports 'hilly', you know the target is located in a non-hilly location.) Implement this functionality in your code. How can you update your search to make use of this extra information? How does your belief state change with these additional observations? Update your search accordingly. Re-do question 4) above in this new environment with the moving target and extra information.

6.1 Updating the agent's belief

At t = 1, the agent's belief of finding the target at each cell will be uniformly distributed. The agent opens a cell randomly, and if it fails to find a target, then the target moves to one of its

neighbors with uniform probability given by $\frac{1}{No.\ of\ neighbours\ of\ current\ target\ location}$

Based on this information, the agent updates its belief in the following way:

<u>Step - 1</u>
We have the following information with us, the prior belief of the agent.

$$P(target\ in\ cell\ i\ @\ t\ |\ observations_t)\ ,\ \ for\ all\ i$$

Step - 2

Now, the target moves to a new location. Since the target can only move to one of its neighbors, at t+1 the target can be in cell i, only if the target was in one of the neighbors of i at t. So, for a cell i, we sum up our belief that the target was in one of its neighbors in the previous step and multiply it with the probability that if the target was one of its neighbors j, with what probability did it move to cell i. The second term is simply $\frac{1}{No.\ of\ neighbours\ of\ i}$.

Incorporating this information, to update the belief:

$$P(\,target\ in\ cell\ i\ @\ t+1 \mid observations_t)$$

$$= \sum_{j\,\in Neighbors(i)} P(\,target\ in\ j\ @\ t \mid observations_t)\, P(\,target\ in\ cell\ i\ @t+1 \mid target\ in\ cell\ j\ @t\,)$$

$$= \sum_{j\,\in Neighbors(i)} P(\,target\ in\ j\ @\ t \mid observations_t)\ *\ \frac{1}{|N_i|}$$

<u>Step - 3</u>

After the target moves to one of its neighbors, the tracker sends the information of the terrain type in which the target is **not** located. Suppose that the tracker returns that T as the terrain for step t+1. This implies that at t+1, the target is not in any cell with terrain type T ($\neg T_{t+1}$). Using this new information, the agent needs to update its prior belief of where the target is. Applying Bayes Theorem:

$$P(\,target\ in\ cell\ i\ @\ t+1 \mid observations_t,\ \neg T_{t+1})$$

$$= \frac{P(\,target\ in\ cell\ i\ @\ t \mid observations_t\,)\ P(\ \neg T_{t+1} \mid observations_t, target\ in\ cell\ i\ @\ t{+}1\ )}{P(\,\neg T_{t+1} \mid observations_t\,)}$$

The first term in the numerator is the prior belief of the target being in cell i. To calculate the second term, we need to consider two cases. If the target is in cell i, and its terrain type is *t* , then

1. the probability of the tracker returning *t* as the terrain type in which the target is not located is 0.
2. the tracker returning $\neg T$, T can be any of the remaining three terrain types. Hence the probability is ⅓

$$\begin{aligned} P(\ \neg T_{t+1} \mid observations_t,\ target\ in\ cell\ i\ @\ t+1\ ) &= 0 \qquad if\ terrain\ type\ of\ cell\ i\ = \neg T \\ &= ⅓ \qquad \text{else} \end{aligned}$$

In the first case, the belief of target being in all cells with terrain Type T becomes 0. Intuitively, this is fairly obvious. If the tracker returns the terrain type T, it implies that the target is not in any cell with terrain type T and hence the agent’s belief for all cells with terrain type T should become 0.

Now, the denominator can be marginalised and re-written as:

$$
\begin{aligned}
&P(\neg T_{t+1} \mid observations_t)\\
&= P(\,target\ in\ cell\ i\ @\ t \mid observations_t)\,P(\,\neg T_{t+1} \mid observations_t,\ target\ in\ cell\ i\ @\ t+1)\\
&\quad + P(\,target\ not\ in\ cell\ i\ @\ t \mid observations_t)\,P(\,\neg T_{t+1} \mid observations_t,\ target\ not\ in\ cell\ i\ @\ t+1)\\
&= Prior\ Belief\ *\ [0\ or\ ⅓]\\
&\qquad + (1 - Prior\ Belief)\ *\ P(\,\neg T_{t+1} \mid observations_t,\ target\ not\ in\ cell\ i\ @\ t+1)
\end{aligned}
$$

The term $P(\,\neg T_{t+1} \mid observations_t,\ target\ not\ in\ cell\ i\ @\ t+1\,)$ is calculated as:
It the tracker returns terrain type T, and the terrain of cell i is not T,

$$P(\,\neg T_{t+1} \mid observations_t,\ target\ not\ in\ cell\ i\ @\ t+1\,) = \frac{|Cells\ not\ of\ Terrain\ Type\ T| - 1}{Total\ cells - 1}$$

The final update turns out to be the following:

$$
\begin{aligned}
&P(\,target\ in\ cell\ i\ @\ t+1 \mid observations_t,\ \neg T_{t+1})\\
&= \frac{PriorBelief \times 1/3}{Prior\ Belief \times 1/3 \quad + \quad (1 - Prior\ Belief) \times \frac{|Cells\ not\ of\ terrain\ type\ T| - 1}{Total\ cells - 1}}
\end{aligned}
$$

<u>**Step - 4**</u>
With the new beliefs, the agent can choose the next cell to search for using rule-1 or rule-2.

Figure 6 shows the total number of searches required by using the decision rules 1 and 2. Rule 2 on an average takes less no. of searches to find the target, in consistence with the results obtained in all other cases.

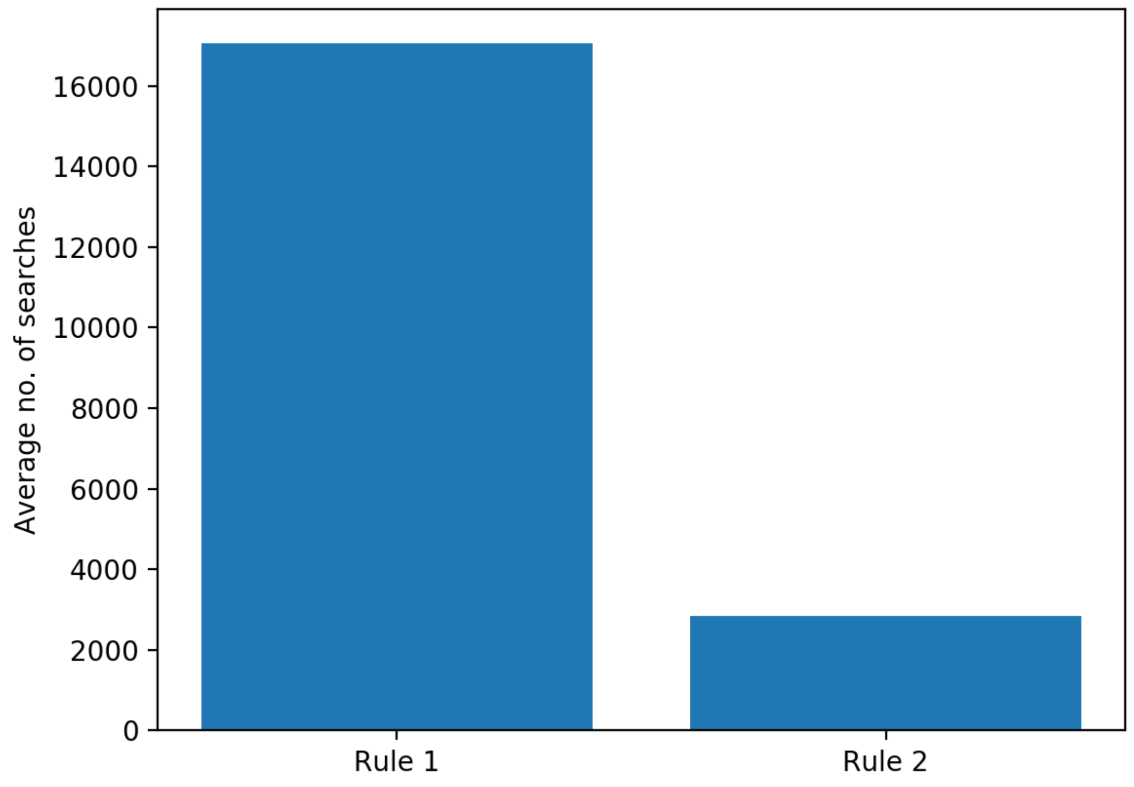


Figure 6: Average number of searches taken by the agent while following Rule 1 and Rule 2. The target is in a map of dimension (50X50). (#simulations = 50)

6.2 Extending Q4 to Moving target environment

In q4, the concept of number of actions comes into the picture. Every move and search incurs a cost of one. The implementation of q4 remains the same for the moving target environment with only difference being in the method of updating the agent's belief. After updating the belief map according to the equation derived in section 6.1, the agent moves and searches the new location based on the decision rules derived in section 4.

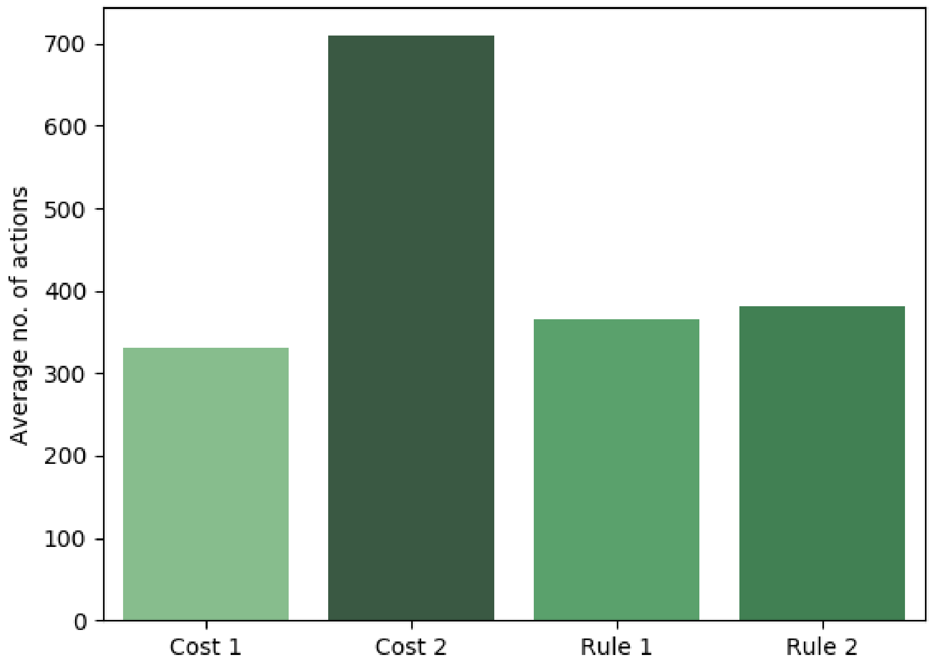


Figure 7: Average number of actions taken by the agent while following Rule 1, Rule 2, One step cost estimation and two step cost estimation. The target is in a map of dimension (10X10). (#simulations = 100)

Figure 7. shows the performance of the four decision rules. In the environment of moving target, since the target is not stationary and the belief of all cells of a particular terrain type becomes zero each time step, it might not be possible to draw conclusions on the decision rules. But one thing on which we can comment is the performance of the two step cost estimate. Out of all four decision rules, two step cost estimate, performs the worst i.e takes maximum actions to find the target. The two step cost estimate, looks two steps into the future to determine the next action. But since in this environment, the target moves after every step, the cost after one step will depend on the new position of the target and hence the utility of looking two steps ahead is lost. More conclusive analysis can be given by taking the environment of higher dimension, but it becomes computationally challenging and hence could not performed.

Appendix

Demonstration of Belief map update for a (5 x 5) environment:

1. Stationary Target
2. Moving Target