CS 520: Introduction to AI
Wes Cowan

Assignment 3 - Probabilistic Search
(November 17, 2019)

197001859: Twisha Naik (tn268)
196003272: Hari Priya (hp467)
197001190: Aditya Lakra (al1247)

Implementation Details

We have a 50x50 map, where each of the location is randomly assigned one of the terrain type from the list below. As the terrain gets more difficult, it is harder to search for it i.e. the more likely it is that even if the target is there, you may not find it (a false negative).

The false negative describes -> P(not finding target at cell i | target is in cell i)

The false negative only depends on the type of terrain.

Terrain Type No.	**Terrain Type Description**	**Probability**	**False Negative Rate (FNR)**
0	Flat	0.2	0.1
1	Hilly	0.3	0.3
2	Forested	0.3	0.7
3	A complex maze of Caves and tunnels	0.2	0.9

Initially, the target is equally likely to be anywhere in the landscape, hence starting out:

- P(Target in Cell$_i$) = 1/ (Total number of cells)

We have a **cell class** where each object has the following attributes:

1. Terrain type -> Integer describing the type of terrain for the particular cell
2. Initial probability of cell being a target without any prior = 1/dim*dim
3. False Negative Rate -> From the table based on the terrain type
4. belief -> P(C$_i$ | Observations$_t$) = Belief[Ci given observations till time t]
5. is_target -> Boolean describing if target is at the particular cell

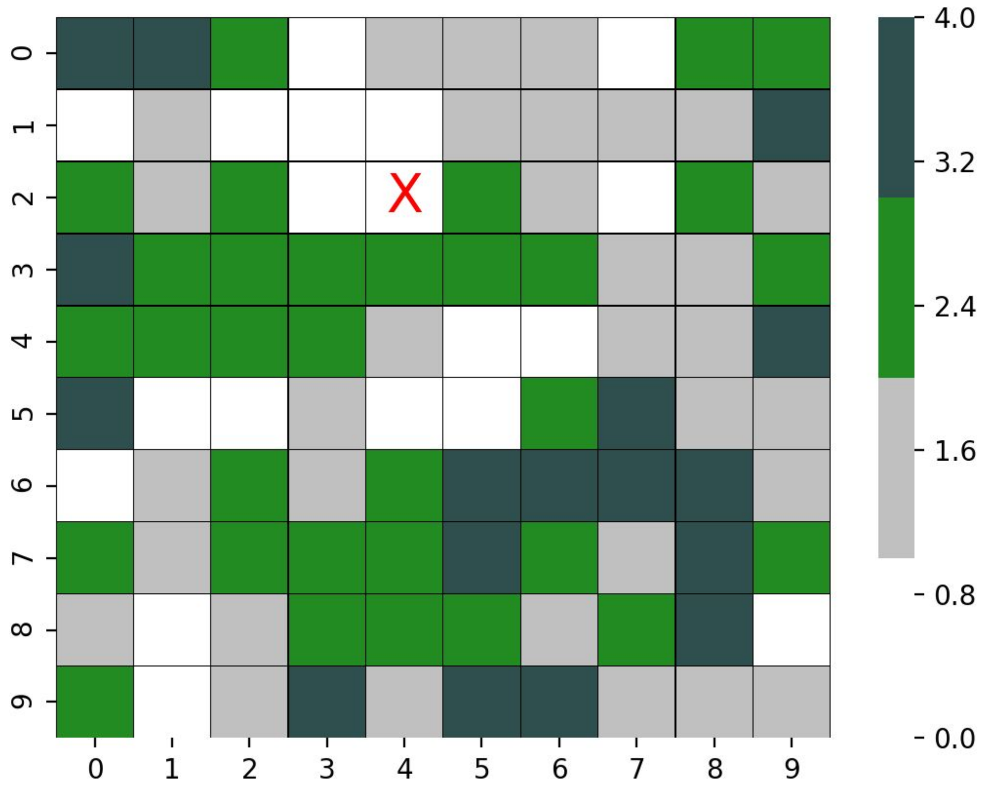


Fig: The sample map with a target and the colors specifying different terrains

Representation Details

1. **P(C_i):** Probability of target in cell i without any prior information
2. **P(Fj):** Probability of finding target at cell j
3. **P(~Fj):** Probability target not found at cell j
4. **Observations$_t$:** Observations through time t **(**Observations will just be a combination of cells explored and target not found)
5. **Belief[C_i] = P (C_i| Observations$_t$):** Belief of the cell at time t
6. **FNR:** False Negative Rate

Questions

1) Update Equations for Belief

Given observations up to time t (Observations$_t$), and a failure searching Cell j (Observations$_{t+1}$ = Observations$_t$ ∧ Failure in Cell$_j$), how can Bayes' theorem be used to efficiently update the belief state, i.e., compute: P(Target in Cell$_i$ |Observations$_t$ ∧ Failure in Cell$_j$).

Upon encountering any failure, the belief of each of the cells need to be updated. Bayes's theorm can be used to update the beliefs by just considering the beliefs at exactly one time step back. This way we do not need to consider the whole sequence failures to get the updated belief.

a. The update for cell j, which is the cell queried and we failed to find a target is as follows:

$$Belief_{t+1}[Cj] = \frac{Belief_t[Cj]*FNR}{Belief_t[Cj]*FNR + \sum_{i\neq j,\, i=1}^{n} Belief_t[Ci]}$$

$$=> Belief_{t+1}[Cj] = \frac{Belief_t[Cj]*FNR}{Belief_t[Cj]*FNR + (1 - Belief_t[Cj])}$$

$$=> Belief_{t+1}[Cj] = \frac{Belief_t[Cj]*FNR}{1 - Belief_t[Cj]*(1 - FNR)}$$

The final equation can be interpreted intuitively as follows:

- Numerator: the prior belief of failure (considering the whole sequence of previous observations) when target exists at Cell j
- Denominator: the prior belief of failure (considering the whole sequence of previous observations) at Cell j

The updated belief is essentially the prior belief of getting the failure when the cell contains the target over the prior belief of getting the failure in all of the cases. In other words, it says out of all the possible failures at cell j what was the probability of failure due to FNR when target is actually present at cell j.

b. The update for cells other than the cell queried (i≠j)

$$Belief_{t+1}[Ci] = \frac{Belief_t[Ci]}{Belief_t[Cj]*FNR + \sum_{i\neq j,\, i=1}^{n} Belief_t[Ci]}$$

$$=> Belief_{t+1}[Ci] = \frac{Belief_t[Ci]}{Belief_t[Cj]*FNR + (1 - Belief_t[Cj])}$$

$$=> Belief_{t+1}[Ci] = \frac{Belief_t[Ci]}{1 - Belief_t[Cj](1 - FNR)}$$

The final equation can be interpreted as follows:

- Numerator: the prior belief that target exists at Cell i (considering the whole sequence of previous observations)
- Denominator: the prior belief of failure (considering the whole sequence of previous observations) at Cell j

The updated belief is essentially the prior belief of cell containing the target over the prior belief of getting failure at the explored cell. Here, we are dividing the belief at time t by a number less than 1. Thus, getting failure at some other cell is increasing the belief of target being at current cell.

Key Observations:

After every failure, the belief of the explored cell is reduced and this results in an increment of the beliefs of all the other cells. This makes sense because since the explored cell gave failure, it weakens the agent's belief that the target is present in that cell. The increment of belief in all the other cells is essentially the weighted distribution (wrt the current beliefs) of the reduction in the belief of the explored cell.

This formula also depicts that, failure on the easier terrain reduces the belief of that particular cell much more than the difficult terrains. In other words, as the FNR for easier terrains is low, failure of finding that cell more confidently says that the target is not present there.

2) Computing probability for finding a target

Given the observations up to time t, the belief state captures the current probability the target is in a given cell. What is the probability that the target will be found in Cell i if it is searched: P(Target found in Cell$_i$|Observations$_t$)?

Probability of finding the target in cell is the product of probability of the target being the cell and the probability of finding the target if it is there.

$$P(Target\,found\ in\ Cell_i \mid Obs_t) = P(Target\ in\ Cell_i \mid Obs_t) * (1 - FNR)$$

and $P(Target\ in\ Cell_i|Observations_t) = Belief[C_i]$

$$\Rightarrow P(Target\,found\ in\ Cell_i \mid Observations_t) = Belief[C_i] * (1 - FNR)$$

Observations:

- Probability of finding the target will always be lesser than the belief of target being at a cell.
- Probability of finding a target in easier terrain is higher than the probability of finding the target in the more difficult terrains.
- This metric gives equal importance to the belief of target being at a location and the ease of finding it.

Example: If a hilly cell has a belief of 0.3, P(Target found) = 0.3*0.7 = 0.21

If a forested cell has a belief of 0.7, P(Target found) = 0.7*0.3 = 0.21

Thus, the probability of finding the target equally depends on both the belief and FNR.

3) Comparison of two 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 rule 1, we create a pool of all the cells with the maximum belief at that point of time and pick a random cell from the pool.

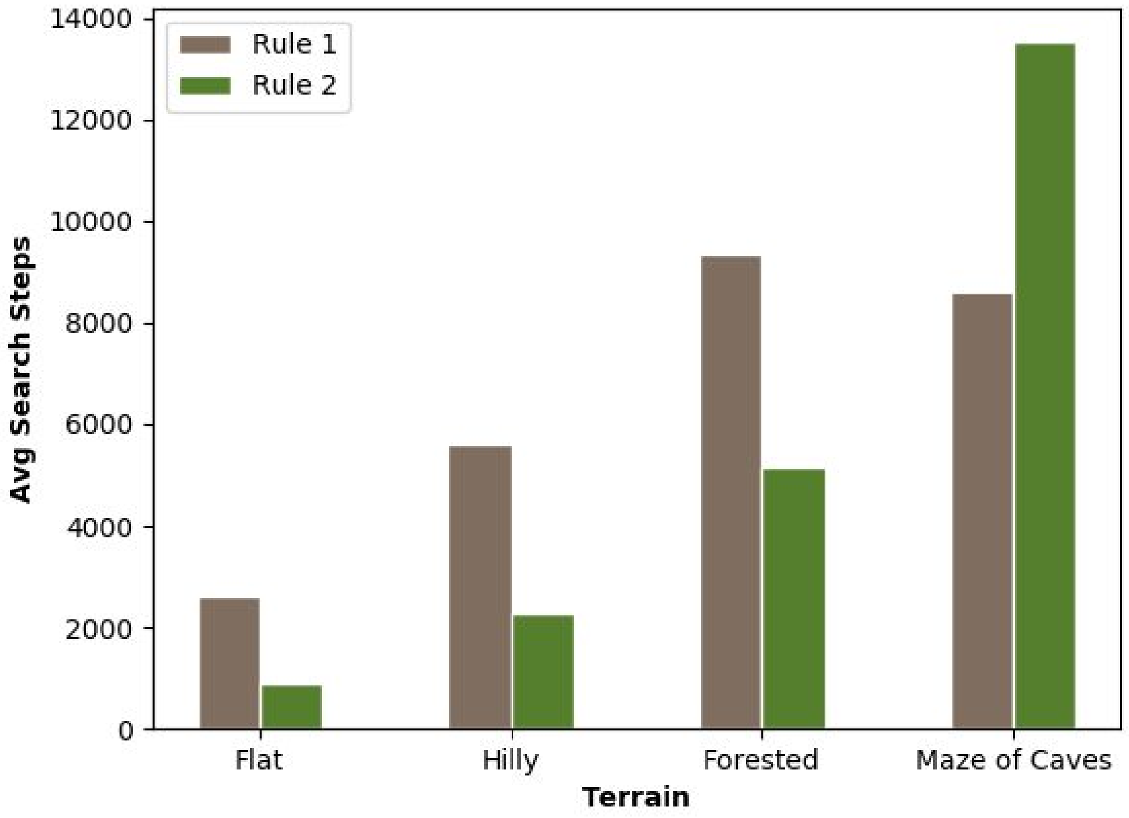
For rule 2, we create a pool of all the cells with the maximum value of the product of belief and the probability of finding the target in that cell (i.e. belief*(1 - FNR)) and then we pick a random cell out of this pool.

Below is the code snippet for this functionality:

```
if rule_no == 1:
       if max_belief == cell.belief:
            max_belief_pool.append(cell)
elif rule_no == 2:
       if max_prob_finding_target==cell.belief*(1-cell.false_negative):
             max_belief_pool.append(cell)
random_index = random.randint(0, len(max_belief_pool)-1)
random_cell = max_belief_pool[random_index]
```

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 less searches), Rule 1 or Rule 2? Why do you think that is?

Rule 2 will always prefer the easier terrain over the more difficult ones and the weighing the belief and (1 - FNR) equally. Due to this, when the target lies in the easier terrain, rule 2 will perform better than the rule 1. Below is the plot for Rule 1 and Rule 2 for a map of dimension 50.

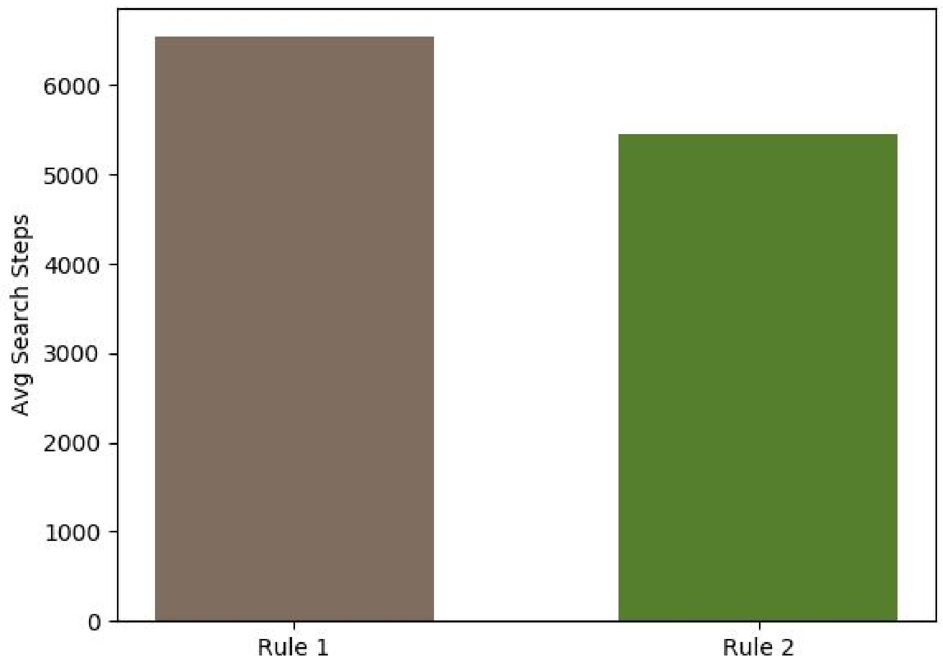


Detailed Analysis:

From the above plot generated by analyzing multiple maps, we can observe that:

- Rule 2 performs better than Rule 1 for the Flat, Hilly and Forested terrains. This is because Rule 2 prioritizes the probability of finding the target in a cell and this value is high for the easier terrains (low false negative rate). So, on average, Rule 2 performs faster when the target is in one of these cells as it searches the cells in the easier terrains first (and more number of times).
- Rule 1 performs better on average than Rule 2 for Maze of Caves which is the most difficult terrain. This is because this terrain has the least probability of finding the target (high false negative rate). So, on average, it performs worse compared to Rule 1 because it tends to keep searching Flat, Hilly and Forested cells before it moves on to the hardest terrain, in spite of the target being in the maze of caves terrain. Hence, it takes more number of search steps to locate the target.

Below is the plot for the average performance of the two Rules over all terrains on a map of dimension 50. Rule 2 performs better than Rule 1 on average.



One more approach was implemented which overcomes the issue we face in the Rule 2 by combining the ideas of both the approaches mentioned in the question.

Proposed Rule 3:

Instead of searching for the cell with maximum probability of finding the cell, we first take the cells with maximum probability of containing the target and then prioritize the cells with easiest terrain i.e. more probability of finding the target.

Below is the analysis for all the three rules for a map of dimension 50 and then over the dimension of 12.

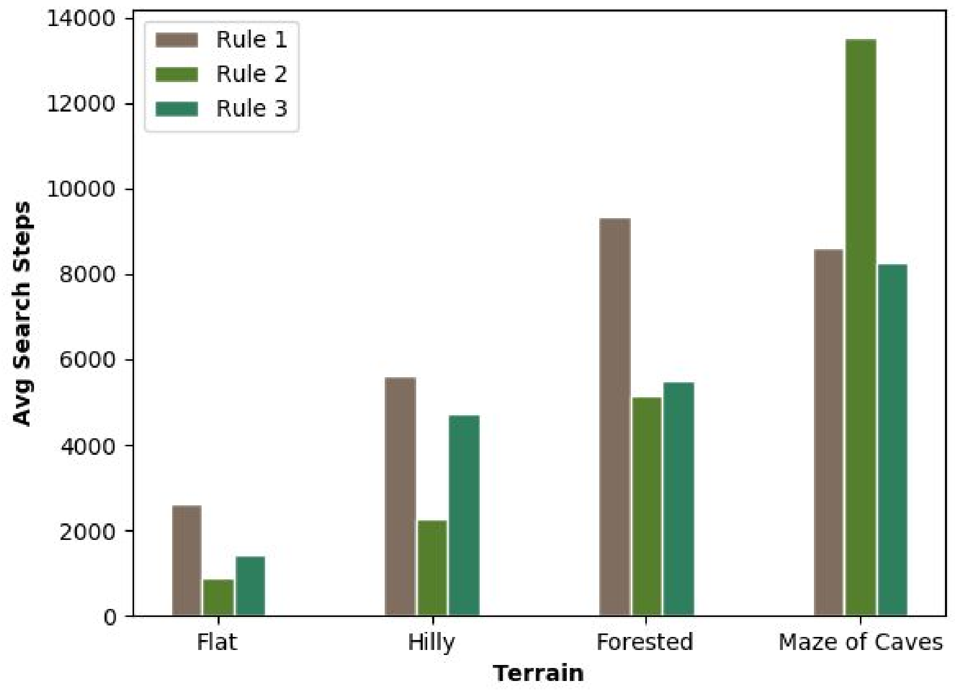


Fig: Analysis for all the three rules for a map of dimension 50x50 - (500 iterations)

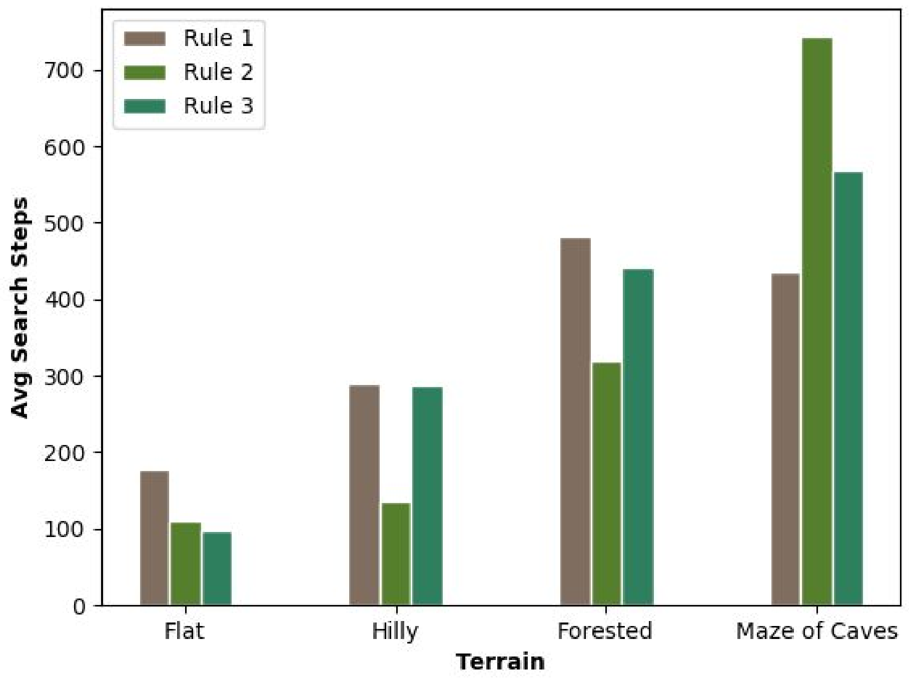


Fig: Analysis for all the three rules for a map of dimension 12x12 - (1000 iterations)

From the plots above, we can see that the performance of the rules is independent of the dimension of the map. Rule 2 performs better than Rule 1 for flat, hilly and forested terrains while it is worse for maze of caves. The reason behind this is as explained in Question 3.

Rule 3 performs worse than Rule 2 in all terrains except the maze of caves. Rule 2 explores cells of the same terrain multiple times starting from the easiest terrain before it moves on to harder terrains while Rule 3 explores the cells once and loops in the same order. So Rule 3 starts exploring the Maze of Caves terrain much quicker than Rule 2. This is advantageous when the target is in the Maze of Caves terrain and hence it is located quickly using Rule 3 compared to Rule 2.

Below is the plot for the average performance of the three rules over all terrains on a map of dimension 50. Rule 3 performs better than both Rule 1 and Rule 2 on average because it never has the worst performance.

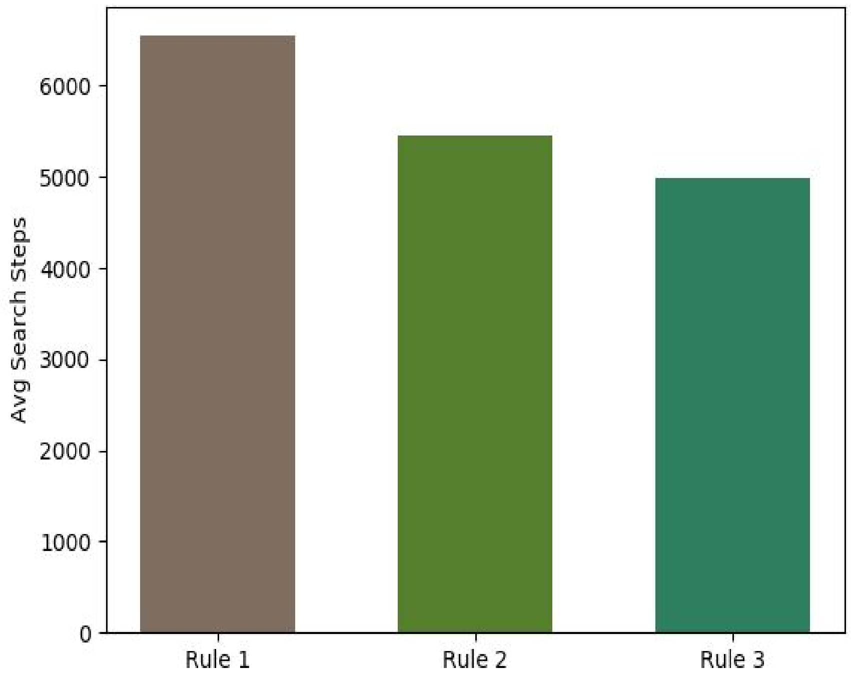
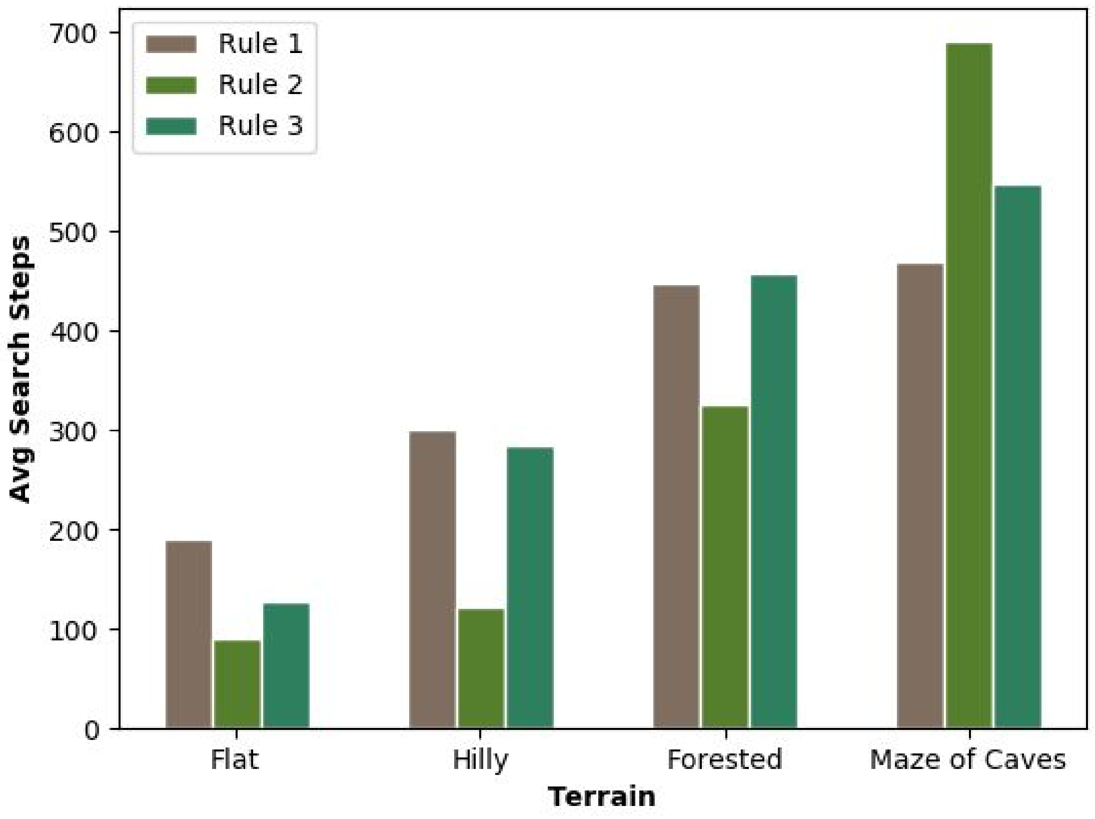
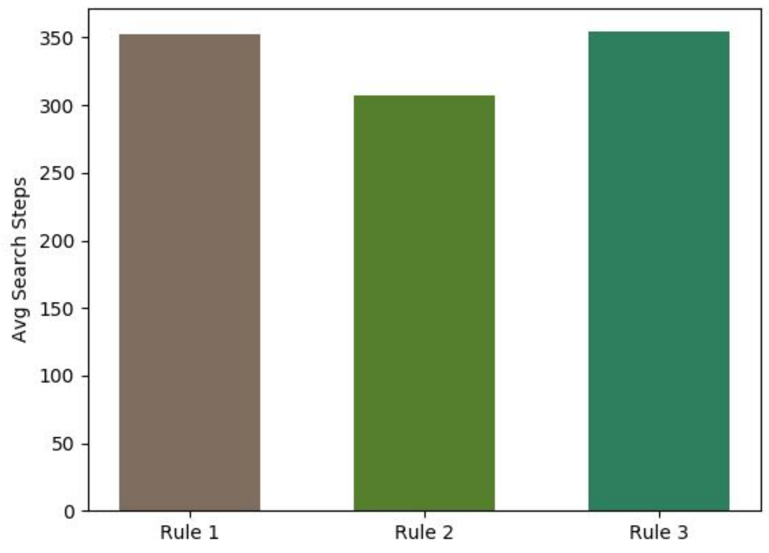


Fig: 50x50 map, 1000 iterations

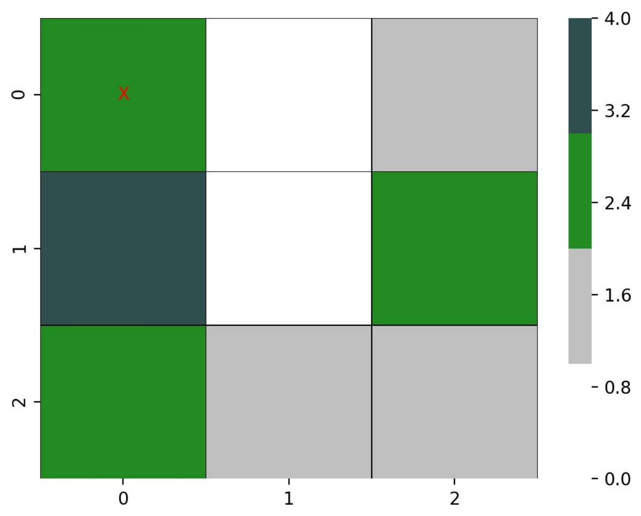
Does that hold across multiple maps?

From the plots given below, it is evident that the results hold even across multiple cell maps.





Analysis of the order in which cells are explored:



Target Location: (0, 0)

Terrain type: 2 (Forested terrain)

Rule 1		Rule 2		Rule 3	
Cell Position	Terrain Type	Cell Position	Terrain Type	Cell Position	Terrain Type
(1,0)	3	(1,1)	0	(0,1)	0
(2,0)	2	(0,1)	0	(1,1)	0
(1,1)	0	(2,2)	1	(0,2)	1
(0,2)	1	(0,2)	1	(2,1)	1
(2,2)	1	(2,1)	1	(2,2)	1
(2,1)	1	(2,0)	2	(0,0)	2
(1,2)	2	(0,0)	2	(1,2)	2
(0,0)	2	(1,2)	2	(2,0)	2
(0,1)	0	(1,2)	2	(1,0)	3
(1,0)	3	(0,0)	2	(1,0)	3
(1,0)	3	(0,2)	1	(1,0)	3
(1,0)	3	(2,1)	1	(1,0)	3
(2,0)	2	(2,0)	2	(0,0)	2

(0,0)	2	(2,2)	1	(1,2)	2
(1,2)	2	(0,0)	2	(2,0)	2
(1,0)	3			(1,0)	3
(1,0)	3			(1,0)	3
(1,0)	3			(1,0)	3
(2,0)	2			(0,0)	2
(0,0)	2				

Key Observations:

- **Rule - 1:** Opens cells randomly whenever the belief is similar for multiple cells. For example, in the first step it chooses the most difficult terrain (terrain type 3).
- **Rule - 2:** It always prefers the easier terrain over the more difficult ones. As seen from the observation, the terrain types chosen are 0s, 1s, 2s, 1s, 2s, 1s, 2s. Before moving to terrain 3 it will iterate over the easier terrains multiple times.

  Here, 0s are not coming again because, the failure on a flat terrain more confidently says says target is not in that cell. Hence, the belief is drastically reduced and coming back to a flat cell where we have failed once, is much more difficult compared to others.

  This rule is not exploring terrain 3 even once.

- **Rule - 3**: It will prefer the cell with highest belief at a particular time. If multiple cells have the same belief, it will prefer easier terrain over the more difficult ones only if both the terrains have equal priority.

  Thus, it ends up exploring the most difficult terrain 3 before making multiple iterations over easier terrain.

4) Modified search using utility function

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.

Approach 1:

One step cost function

For making the modification, we define a cost function which describes the cost of exploring a cell given a current location. Cost function will give me the expected number of actions needed to find the target. Suppose, the current location of agent is cell i = (x_i, y_i).

$$\boxed{Cost[i, j] = \frac{1}{p} + manhattan\_distance(cell_i, cell_j)}$$

The term 1/p gives the expected number of trials needed to reveal the cell if that cell is repeatedly queried. Hence, it is like an estimate of what the cost would be if we go from cell i to j and then remain there until we find the target.

Here, based on the definition of p, we have coded two different cost functions.

1. Cost based on Rule-1: p = Belief[Cj]
2. Cost based on Rule-2: p = Belief[Cj] * (1 - FNR)

We pick the cell j which has the minimum expected cost. This is the greedy approach where we pick the cell where we can reach with minimum cost.

Augmented cost function (Two step function where average estimated cost is taken at second step)

To estimate the cost in a better way we take the average cost of the entire board considering the new cell as the center. This is like looking at one step in the future before deciding the next step.

X -> A -> (average cost of going to any other cell B)

This will take care of the following things:

- It will prefer the cell which has a neighborhood with higher probability of finding the target.
- It will also prefer the cells towards the center of the map over the corner of the map as going anywhere on the map from the corner is costlier than going to any cell from the center.

$$\boxed{Augmented\ Cost[i,\ j]\ =\ \frac{1}{p} + \ manhattan\_distance(cell_i,\ cell_j)\ + \frac{1}{n}\sum_{k=1}^{n} Cost[j,\ k]}$$

Again, based on the value of p, we have coded two different utility functions with the augmented cost. And the cost is calculated based on the previous definition of cost (i.e the one step estimate)

1. Cost based on Rule-1: p = Belief[Cj]
2. Cost based on Rule-2: p = Belief[Cj] * (1 - FNR)

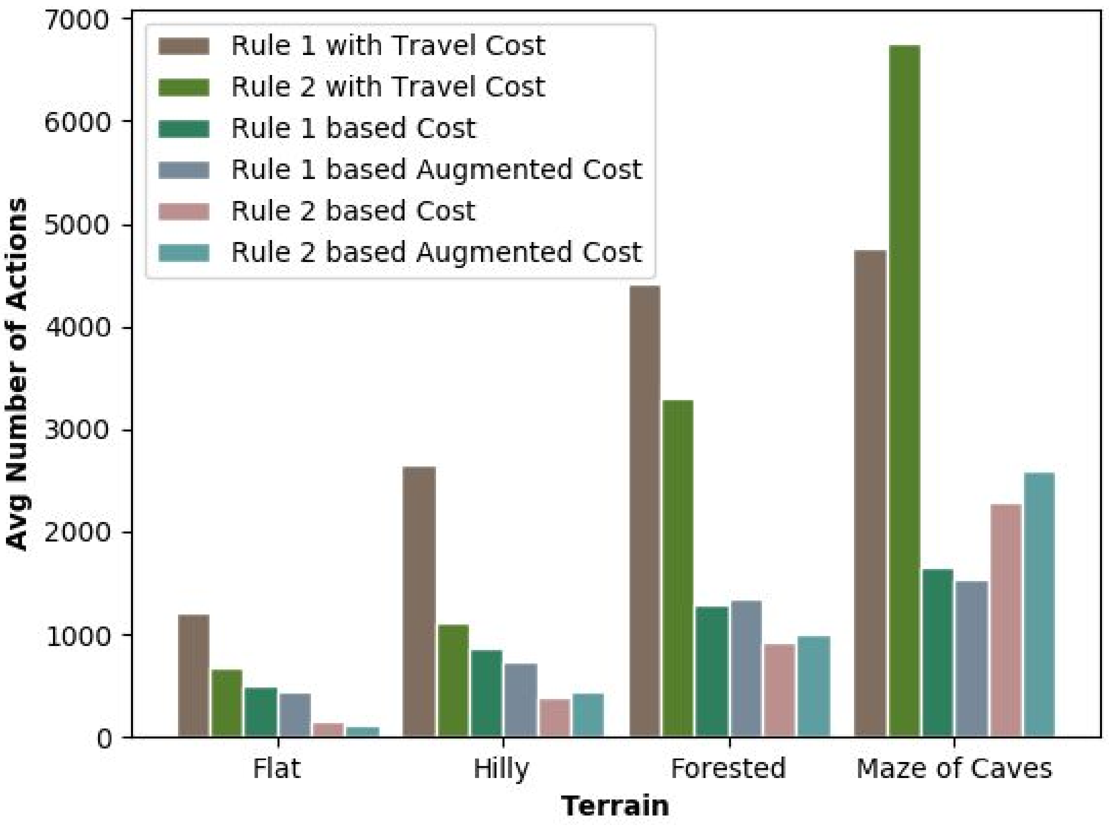
In the previous approach we were stopping at just expected cost of cell j. Now we take a further look at the expected cost of the entire map from the cell j.

Comment:

In the definition of cost, we have assumed 1/p to be the expected number of trials needed to reveal the cell given that we query the cell repeatedly. Hence, rule 2 should better justify the definition.

Analysis

We have plotted the number of steps taken by using Rule-1 and Rule-2 with Travel Cost and after that including the Cost and Augmented Cost functions. (Cell map size = 12x12)



Detailed Analysis:

- The performance with Rule 1 and Rule 2 based Cost functions is always better than the performance of simple Rule 1 and Rule 2 with the consideration of Travel Costs. This is because without inclusion of cost function, we simply choose the cell with highest belief disregarding the cost that will be incurred due to travel. The cost function chooses the cell to be explored that will minimize the number of actions required to search the target.
- In other words, the cost function might prefer a nearby cell with lower belief over a farther cell with high belief.
- The comparison of performance of Rule 1 and Rule 2 with and without travel costs remains the same. Rule 2 is better in Flat, Hilly and Forested terrain whereas Rule 1 is better in the Maze of caves.

- In all cases where Rule 2 with Travel Cost performs better than Rule 1 with Travel Cost, it also implies that Rule 2 based Cost performs better than Rule 1 based Cost in these cases and vice-versa.

  Rule 2 performs better than Rule 1 for the Flat, Hilly and Forested Terrains while Rule 1 performs better than Rule 2 for the Maze of Caves. This might be the case because the cost function for Rule 2 also considers the probability of finding the target in a cell besides the belief of a cell containing the target.

  The estimated number of times a cell needs to be explored to find the target with:

  Rule 1: 1/cell.belief

  Rule 2: 1/(cell.belief * (1-FNR(cell)))

  This means that Rule 2 starts exploring cells in easier terrains first and keeps moving to the harder terrains. This results in Rule 2 exploring the easier terrains like Flat, Hilly and Forested terrains much quicker than Rule 1. So, the target is found much faster with Rule 2 compared to Rule 1.

  But even when the target is in Maze of Caves, Rule 2 still explores this terrain at the end after exploring all the other terrains and hence takes more number of action steps. However, in this case, Rule 1 only explores cells based on the belief of a particular cell containing the target and hence, it explores Maze of Caves terrain before Rule 2 and has a greater chance of finding the target quickly compared to Rule 2.

- The performance of the rules with Augmented Cost function is comparable to the performance of the rules with Cost function. The augmented cost is not significantly better than the normal one step estimator.

To remove the bias towards the center cells, we propose another approach in the two step estimation.

Approach 2:

$$\boxed{Augmented\ CostWithMin[i, j] = \frac{1}{p} + manhattan\_distance(cell_i,\ cell_j) + min_k\{Cost[j,\ k]\}}$$

Again, based on the value of p, we have coded two different utility functions with the augmented cost. And the cost is calculated based on the previous definition of cost (i.e the one step estimate)

This performs better than the average being used at second step.

The average number of steps on 12x12 grid with augmented cost using average and using max.

Augmented cost with Min		**Augmented cost with Average**	
Rule 1	**Rule 2**	**Rule 1**	**Rule 2**
1084	879.3	1304	1026

Approach 3:

$$Cost[i, j] = manhattan\_distance(cell_i, cell_j) + 1 + (1 - belief(cell_j \mid NF\ at\ i)) * \\ min_k \{ manhattan\_distance(cell_j, cell_k) + 1 + (1 - belief(cell_k \mid NF\ at\ i, NF\ at\ j)) \\ \}$$

Note: NF denotes not found in the above equation.

In the second approach, we have used a different cost function. Whenever we do not find target at cell i, we calculate the cost of all the cells on the map and randomly choose the ones with the minimum cost.

To calculate the cost after failure at cell-i we consider the following:

1. First step cost:
    a. Cost of moving to a new cell-j
    b. Cost of querying cell-j
2. Probability of failure at cell-j given failure at cell-i
    a. If target found - no additional cost would be added
    b. If target not found - more explorations are required
3. Future steps cost:
    a. Min of all discounted future explorations
4. Future explorations are discounted by the belief of failure at cell-j given failure at cell-i

Total cost = First step cost + (discounting factor) * Future steps cost

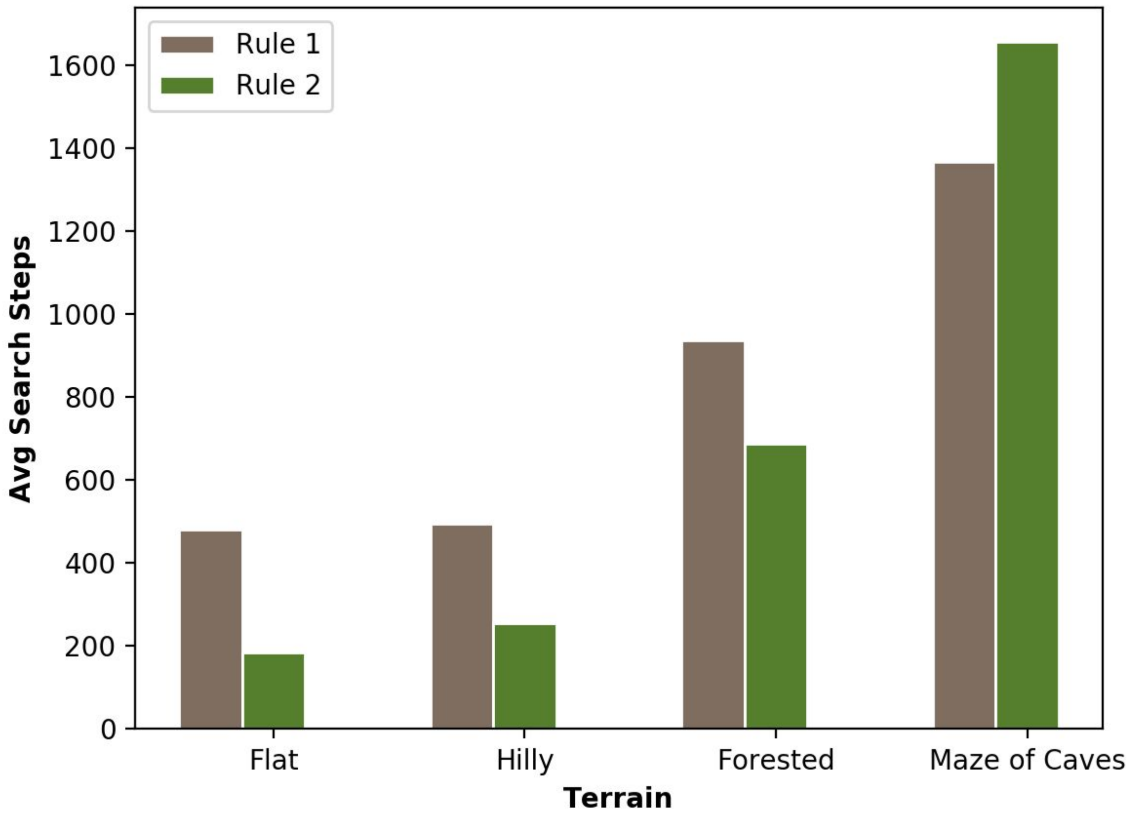
Discounting factor = belief of failure at cell-j given failure at cell-i.

Discounting factor is required to include the case when we might end up finding the target at cell-j. This is required because the probability of future explorations will be incurred only when we fail at cell j.

To calculate the future cost, keeping the computational cost in mind, we just go one more step ahead in the future. For future steps cost we take the sum of the following:

1. Cost of moving to a new cell-k
2. Cost of querying cell-k
3. Probability of not finding the target at cell-k

The performance of Rule 1 and Rule 2 using Approach 2 is as observed from the plot below on a 12x12 map.



Here, it is always choosing the neighbor with the highest belief. The algorithm is not even going 2,3 steps away to explore a better cell.

5) 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.

Over here the drunk guy is just considering the value of (1 - FNR) to choose which locations to search for. If the key is actually present in one of the places which have the least (1 - FNR) value, the drunk guy's method would still be effective. However, if the key is not present in one of the places which have the least (1 - FNR) value, the drunk guy would never be able to find his keys. This is because he would never look at places difficult to search.

The Rule 2 approach can be considered as an improvement of the drunk guy's algorithm. In Rule 2 we not only consider the easier terrain but we also factor in the belief of target being at a given cell. Rule 2 would perform worse than Rule 1 in the cases when the target is present in a hard to find location. But unlike the drunk guy's algorithm, Rule 2 also relies on the belief of a cell containing the target. With continuous failures on the easy to explore cells - which would lead to a reduction in their belief values - the agent will eventually start exploring the hard to explore cells.

While we are discussing the best location to look for given the beliefs, let's also consider the proposed Rule 3. In that case, we would prefer the easier terrains only when multiple cells with different terrain have the same belief.

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.)

Pseudo-code for this problem statement:

```
Generate a cell map with different terrains
Set target at a location
Initial belief of each cell = 1/total cells

while (target not found):
        Get target terrain information from the tracker
        Update belief of each cell based on the information from tracker

        Query cell with the highest belief
        If target found:
                Break out of the loop

        Update belief of each cell based on the failure of queried cell

        Move the target to one of the neighboring cells
        Update belief of each cell as the target has moved using beliefs of
its neighbors
```

As seen from the pseudo code, 3 different update equations are required now.

Splitting the problem in different steps

Step 1 - Initial knowledge at each time t

$$\boxed{P(in\ i\ @\ t \mid obs_t) \text{ -- for all i}}$$

Step 2 - Let the target move

The target can move in one of the neighboring cells with equal probability.

Idea: For the target to be in cell i at time t+1, it has to be in one of its neighboring cells at time t. Thus, we marginalise the probability over the neighbors of the cell at the previous time step.

$$\boxed{P(in\ i\ @\ t+1 \mid obs_t) = \sum_{j \in N(i)} P(in\ j\ @\ t \mid obs_t) \cdot P(in\ i\ @\ t+1 \mid in\ j\ @t)}$$

1. $P(in\ j\ @\ t \mid obs_t)$ -- this is known from the state of board at time t
2. $P(in\ i\ @\ t+1 \mid in\ j\ @t) = \frac{1}{Number\ of\ neighbors\ of\ j}$ -- this is because if the target is at a particular cell, it can move to all of its neighbors with equal probability

Step 3 - Receive ~T_{t+1} from the tracker

$$\boxed{P(in\ i\ @\ t+1 \mid obs_t,\ \neg T_{t+1}) = \frac{P(in\ i\ @\ t{+}1 \mid obs_t) \cdot P(\neg T_{t+1} \mid obs_t,\ in\ i\ @\ t{+}1)}{P(\neg T_{t+1} \mid obs_t)}}$$

1. $P(in\ i\ @\ t+1 \mid obs_t)$ -- this is known from step-1
2. $P(\neg T_{t+1} \mid obs_t,\ in\ i\ @\ t+1) = P(\neg T_{t+1} \mid in\ i\ @\ t+1)$
   a. If Terrain(i) == $T_{t+1} \rightarrow 0$
   b. If Terrain(i) != $T_{t+1} \rightarrow ⅓$: This is because we are assuming that the tracker will return one of the terrain the target is not at with equal probability.

3. By marginalizing the probability over target at i and target not at i,

$$P(\neg T_{t+1} | obs_t)$$
$$= P(in\ i\ @t+1 \mid obs_t) \cdot P(\neg T_{t+1} | obs_t,\ in\ i\ @t+1) + P(not\ in\ i\ @\ t+1 \mid obs_t) \cdot P(\neg T_{t+1} | obs_t,\ not\ in\ i\ @t+1)$$

- Now, $P(not\ in\ i\ @\ t+1 \mid obs_t) = 1 - P(in\ i\ @\ t+1 \mid obs_t)$
- $P(in\ i\ @\ t+1 \mid obs_t)$ -- we know from step-1
- Only thing left to calculate is $P(\neg T_{t+1} | obs_t,\ not\ in\ i\ @t+1)$

a. If Terrain(i) == $T_{t+1} \rightarrow \frac{1}{3} \cdot \frac{Number\ of\ cells\ not\ having\ T_{t+1}}{Total\ cells - 1}$

b. If Terrain(i) != $T_{t+1} \rightarrow \frac{1}{3} \cdot \frac{Number\ of\ cells\ not\ having\ T_{t+1} - 1}{Total\ cells - 1}$

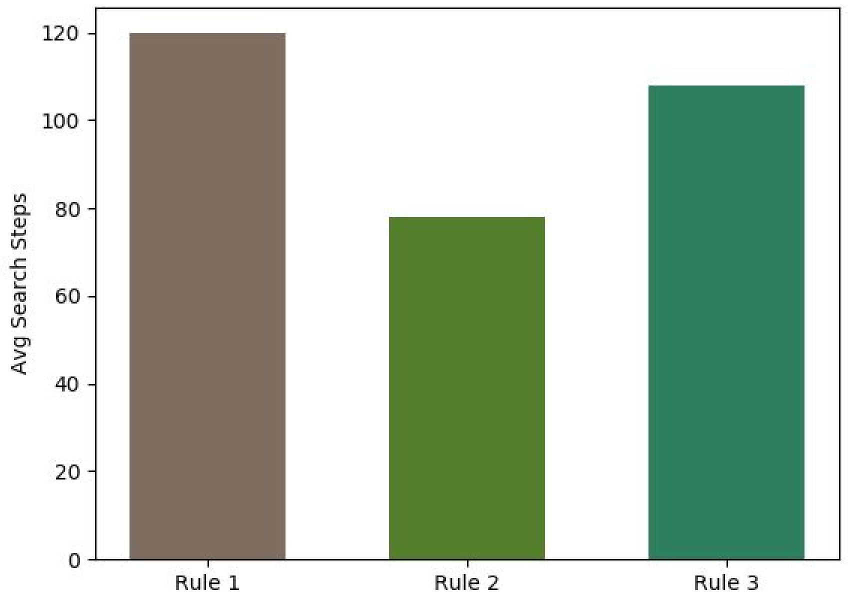
Step 4 - Query cell with maximum belief and target not found

$$\boxed{P(in\ i\ @\ t+1 \mid obs_t,\ \neg T_{t+1},\ fail\ @\ t+1) = \frac{P(in\ i\ @\ t{+}1 \mid obs_t, \neg T_{t+1}) \cdot P(fail\ @\ t{+}1 \mid obs_t, \neg T_{t+1}, in\ i\ @\ t{+}1)}{P(fail\ @\ t{+}1 \mid obs_t, \neg T_{t+1})}}$$

This is the update we were performing in the normal cases (the stationary target) because the only new information added is the failure at the new cell. For equations refer the section 1.

The plots and the analysis on 12x12 map for 1000 iterations can be found below.

Below is the plot for all the three rules while considering the moving target on a 12x12 map for 1000 iterations.



Analysis:

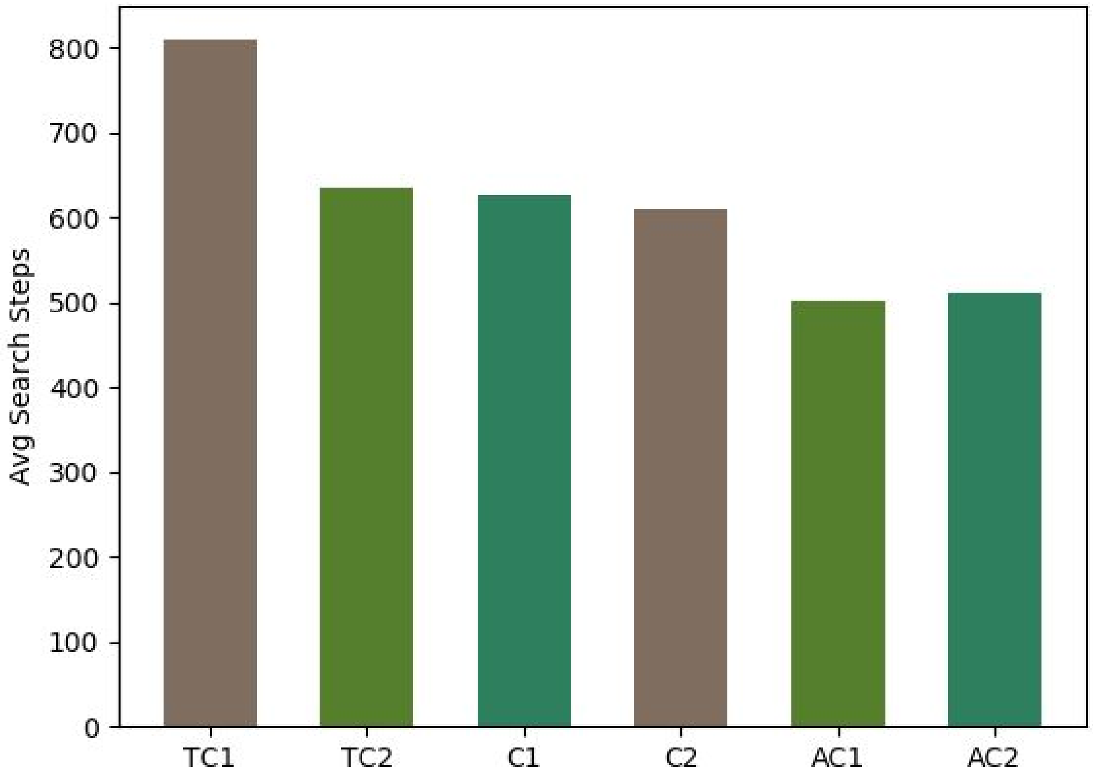
- The number of steps needed to find the target now are much lesser as compared to the normal stationary target case. This is because even though the target is moving we are getting additional information which helps us locate the target faster.
- This is a 12x12 map and as seen from the plot, the average number of steps needed to find the target is now less than 144, which means we are not iterating the entire map even once. In fact it is almost half in case of Rule 2.
- Even having the information about where the target is not located helps us finding the target faster.
- Suppose, the tracker was working fine and we always get the correct terrain of the target. This would drastically improve our performance. And on the other hand, if we did not have the tracker at all, it would be much more difficult to find the target compared to the normal stationary target case.

Comment:

Terrain wise analysis does not give any more insights here as our target is moving.

The analysis will depend on the terrain type of neighbors → basically, the entire cell map.

By including Rule 1 and Rule 2 with Travel Cost (TC1, TC2) , Cost function (C1, C2) and Augmented Cost function (AC1, AC2) from the question 4, the performance observed is as below. (12x12 map and 1000 iterations)



- The inclusion of utility functions is improving the performance of the normal Rule 1 and Rule 2.
- Adding the cost function described in approach 1 of part 4 is giving better results and the augmented cost with the average is performing even better.