CHALLENGES

In this project I have implemented Reinforcement learning algorithm, Markov Decision Process, Dynamic Programming to find optimal CS. The detailed implementation is explained below:-

Spatial domain is the domain which looks at the distance between points as features. For this I have used map images to extract spatial features.

Challenge no1:-

Initial challenges were that now a normal gray scale image 1000*1000 will have 1000,000 points on it and working with so many points will be computationally heavy.

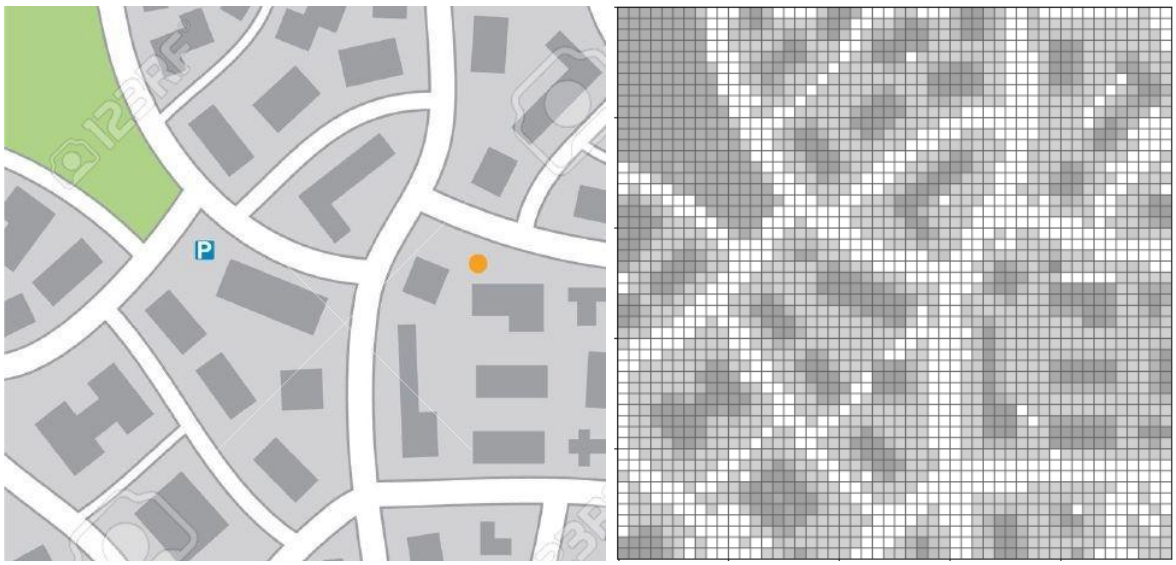


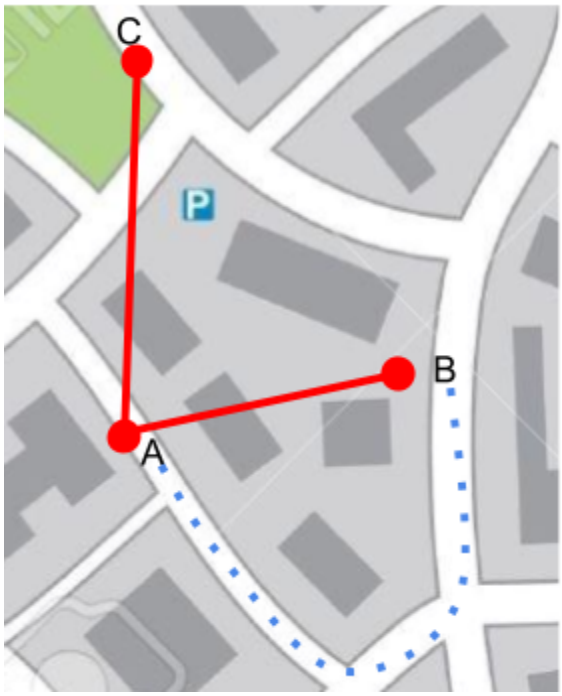
Fig: original image , preprocessed image

So I first preprocessed the image, in which I used tiles of 20*20 and divide image into grid of 50*50. For each grid cell I filled with the mean value of tile. For example for coordinate (1,1) in 50*50, I filled it with mean of the tile which represents this

coordinate in 1000*1000 images. In this way our data became much simpler to deal with.

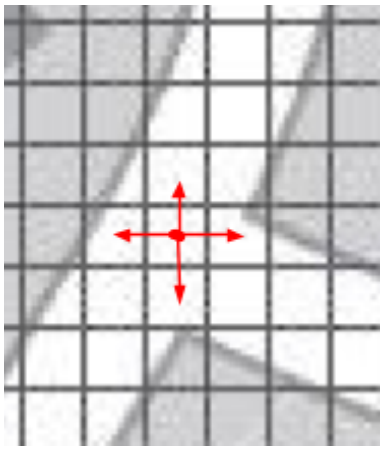
Challenge no2:-

Now the spatial feature alone was not able to give the correct result because, If I simply calculated the distance between locations A,B,C then A and B is closer than A and C but in reality between A and B there is a longer path.



So somehow we have to give the information of path(i.e. roads) and non path(like houses, apartments etc). So for this we used Reinforcement Learning. RL is a reward and punishment based algorithm in which the agent learns about its environment by interacting with it and receiving the reward. An agent always tries to achieve positive reward and will try to minimize negative reward. Here each cell of grid is a state in which our agent can be and all the cells falling on the valid paths i.e. roads we assigned reward of -1 and for all non-path points we assigned reward

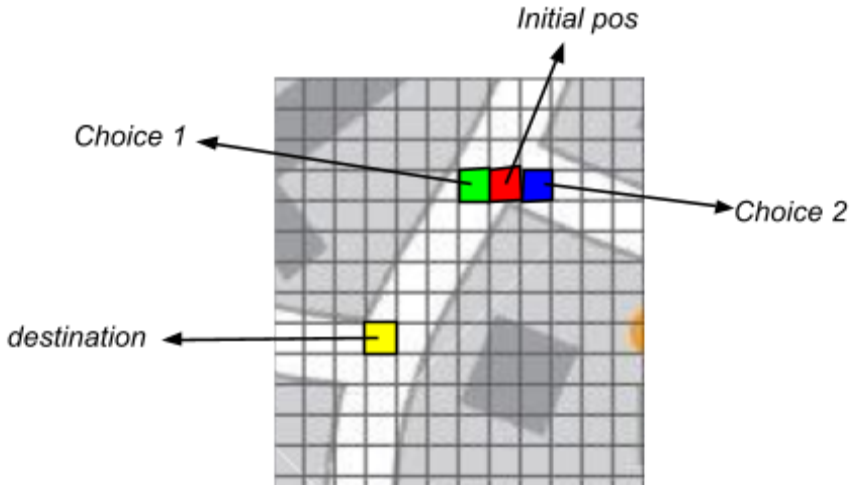
of -10 in the 50*50 grid. From a given state an agent can move either Left/Right/Up/Down. So the actions are L/R/U/D. Thus we have defined all the parameters of our RL system: States, Actions, Reward.



Now due to this our agent will try to avoid houses as its award is -10. Note:- we assigned reward for valid paths -1 and not any positive or 0 value because travelling on the valid path is not our main goal. Our main goal is to reach our destination as fast as we can, so a -1 reward will prevent agents from just roaming .

Challenge no3:-

Next issue is that not all good choices are good choices in the long run. To illustrate this here is an example.



Here both Choice1 and Choice2 are on the valid path so their reward is -1 each but we can see Choice2 is not a good choice in the long run as maximum reward will be when we follow Choice 1. Thus a better decision can be made by considering the long-term impact of our decision. To overcome this issue we will define two types of rewards.

- Short term reward a.k.a reward
- Long term reward a.ka. Return or value function

$$v_\pi(s) \doteq \mathbb{E}_\pi\left[G_{tt} = s\right] = \mathbb{E}_\pi \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, \text{ for all } s \in \mathcal{S}$$

Long term reward is the sum of all rewards from that step. Thus the main goal of our agent is the maximum total return. So now we will calculate all the future rewards of a step and based on that we will make decisions for example, Let in above example return of green cell is -7 while return of blue cell is -10.

Challenge no4:-

Introducing the Value function helped us to avoid issues of getting caught in local optimal decisions but it created another problem. Now Let's see how this value

function is calculated. This value function is the sum over all the future rewards an agent can get starting from the current state till the final state. But finding all future rewards is not easy as there are many choices of steps agents can take from their current state to reach their destination. Like in the above example, from the blue cell to the final yellow cell there are thousands of choices. Calculating the value function of all these paths is not possible.

For this we use the Bellman equation for state-value and action value function. These equations allow us to write the value function of a state in terms of the value function of it's next terms only, without waiting to observe all the future rewards.

Two equations derived from bellman equation are as follows:

State-value function

$$
\begin{aligned}
v_\pi(s) &\doteq \mathbb{E}_\pi\left[G_t \mid S_t = s\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s\right] \\
&= \sum_a \pi(a \mid s) \sum_{s'} \sum_r p\left(s', r \mid s, a\right)\left[r + \gamma \mathbb{E}_\pi\left[G_{t+1 t+1} = s'\right]\right] \\
&= \sum_a \pi(a \mid s) \sum_{s', r} p\left(s', r \mid s, a\right)\left[r +_\pi \left(s'\right)\right], \qquad \text{for all } s \in \mathcal{S},
\end{aligned}
$$

Action-value function

$$
\begin{aligned}
q_*(s, a) &= \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_*\left(S_{t+1}, a'\right) \mid S_t = s, A_t = a\right] \\
&= \sum_{s', r} p\left(s', r \mid s, a\right)\left[r + \gamma \max_{a'} q_*\left(s', a'\right)\right]
\end{aligned}
$$

This will allow us to make optimal decisions. Here we will have k linear equations in k variables where these k variables are the value function of k states.

Challenge no5:-

Now when k is small like 3 or 5 solving systems of linear equations is easy and can be easily solved using matrix multiplication. But in most of the case no of states are huge like in our case no of states is 50*50 that is 2500 states. So solving 2500 equations is computationally heavy. So for this we will use Dynamic programming in which we will iteratively calculate the value function of k states. The initial approximation for value function of each states is chosen arbitrarily (except that the terminal state, if any, must be given value 0), and each successive approximation is obtained by using the Bellman equation as an update rule:

$$
\begin{aligned}
v_{k+1}(s) &\doteq \mathbb{E}_{\pi}\left[R_{t+1} +_k \left(S_{t+1}\right) \mid S_t = s\right] \\
&= \sum_a \pi(a \mid s) \sum_{s',r} p\left(s', r \mid s, a\right)\left[r + \gamma v_k\left(s'\right)\right]
\end{aligned}
$$

Also, initially we will assign a uniform random policy for each state i.e. 1/4 because there are 4 actions L/R/U/D. When we iteratively calculate the value function then it will converge to some value for the current policy. This step is called Policy evaluation. Then for this value function we will make greedy choices for each state i.e. highest probability will be given to that action which will give us maximum return/cumulative reward. This step is called Policy Improvement. Now our Value function no more represents the current Policy so we will again start calculating value functions under this new Policy until we converge. We will keep repeating these two steps policy evaluation and Policy Improvement one after another until the Current Policy is optimal to its own value function. Then this value function and Policy will be global optimal.

I have implemented this algorithm from scratch using python, numpy and for plotting graphs and maps I have used opencv and matplotlib.