Homework #2

Masafumi Endo, M.S. Student in Robotics
ROB534 - Sequential Decision Making in Robotics
Oregon State University

February 15, 2021

Questions

a

The performance guarantee relative to optimal for the greedy solution is expressed as:

$$F(\mathcal{A}_{greedy}) \geq (1 - \frac{1}{e}) \max_{|\mathcal{A}| \leq K} F(\mathcal{A}), \tag{1}$$

where K represents a number of sensors. Equation 1 expresses that the returned $\mathcal{A}_{greedy}$ by the greedy solution has at least ˜63% performance compared to the optimal solution that is NP-hard to achieve. Hence, it can be said that the greedy algorithm gives near-optimal solution.

b

The performance of the cost-benefit greedy algorithm that maximizes the total quantity $F(\mathcal{P})/C(\mathcal{P})$ could not be guaranteed for any K samples along the trajectory followed by the vehicle. The reason is that $F(\mathcal{P})$ divided by $C(\mathcal{P})$ could not maintain its submodularity due to its non-monotonic nature, while both $F(\mathcal{P})$ and $C(\mathcal{P})$ are submodular functions.
This approach would perform poorly if the robot has to explore environments where the information spatially varies, and there are no correlations. In such a case, the cost function will become more complex, but we could not model it exactly since it never knows the environment's model before exploration. It will happen when the robot aims to find paths in (partially) unknown environments such as lake monitoring and exploration on rough terrain.

Programming Assignment

Step 1

i

Algorithm 1 shows the overview of the greedy solver and Algorithm 2 shows the description of the function named GetGreedyAction. There are two action selectors named GetGreedyAction and GetShortestAction. The former one is used for obtaining action that maximizes information gain, and the later one is used for obtaining action to reach the goal position after stopping greedy information gathering. There are two networks named WorldEstimationNetwork that estimates what the world looks like given the explored map and DigitClassificationNetwork that takes the estimated world and provides an estimate of what digit the world belongs to.

Algorithm 1 Greedy exploration

```
flag_greedy = true
while robot does not reach goal do
    if flag_greedy = true then
        a = GetGreedyAction(map_exp)
    else
        a = GetShortestAction(goal)
    end if
    map_exp = UpdateMap(a, map_exp)
    map_est = WorldEstimationNetwork(map_exp)
    digit_est = DigitClassificationNetwork(map_est)
    if digit_est = digit then
        if p_{digit_est}(s, map_est) > p_th then
            flag_greedy = false
        end if
    end if
end while
```

Algorithm 2 GetGreedyAction(map_{exp})

map_{est} = WorldEstimationNetwork(map_{exp})
$H(s, map_{est}) = \sum_{i=0}^{9} -p_i(s, map_{est}) \log_2 p_i(s, map_{est})$
for $a \in \mathcal{A}$ **do**
  $map_{hal} = map_{exp}$+Observation(map_{est}, a)
  map'_{est} = WorldEstimationNetwork(map_{hal})
  $H(s', map'_{est}) = \sum_{i=0}^{9} -p_i(s', map'_{est}) \log_2 p_i(s', map'_{est})$
  $IG = H(s, map_{est}) - H(s', map'_{est})$
end for
$a^* = \arg\max_{a \in \mathcal{A}}(IG)$

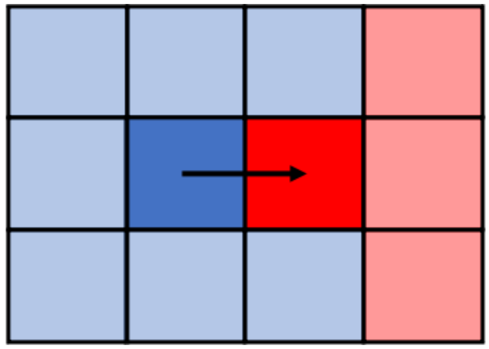


Figure 1: Example of the exploration and map update w/ estimated world. The dark and light blue represent the current robot's location and corresponding explored map. The dark and light red represent the next robot's location and corresponding explored map. Note that the light red regions aren't actually explored but estimated by the neural network.

GetGreedyAction takes explored map and selects the best action a^* that maximizes information gain from $a \in \mathcal{A}$. To do so, it first estimates the current world and classifies the world based on the estimated world with probabilities. The entropy of the current state is calculated as,

$$H(s, map_{est}) = \sum_{i=0}^{9} -p_i(s, map_{est}) \log_2 p_i(s, map_{est}), \tag{2}$$

where s, map_{est} represent the current state and estimated map and p represents the probability of each digit given by classification network. Then, it assumes that the robot takes the possible action and calculates the entropy of the next state. Note that the robot could not explore the frontier of one-step lookahead before taking action so it hallucinates the frontier based on map_{est}, as shown in Fig. 1.
WorldEstimationNetwork then takes map_{hal} and estimates the world, and the estimation result map'_{est} is used for digit classification. After classifying the digit, the entropy of the possible next state is calculated as,

$$H(s', map'_{est}) = \sum_{i=0}^{9} -p_i(s', map'_{est}) \log_2 p_i(s', map'_{est}), \tag{3}$$

where s', map'_{est} represent the state based on a' and estimated map and p represents the probability of each digit given by classification network. The information gain IG is then calculated with Eq. 2 and 3 as,

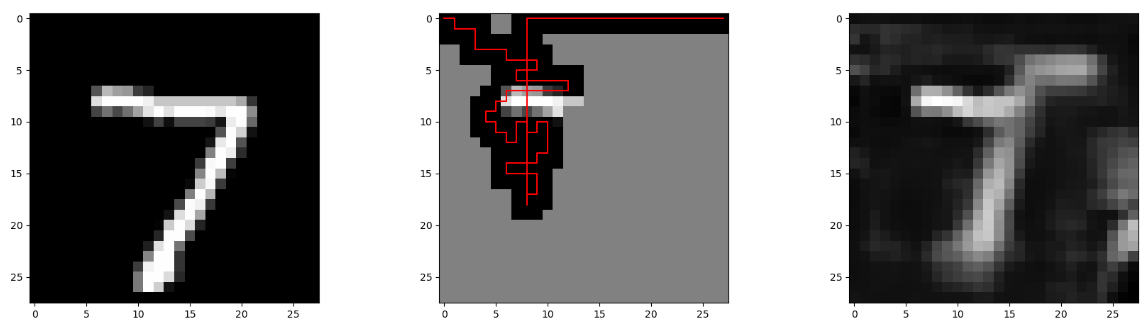
$$IG = H(s, map_{est}) - H(s', map'_{est}) \tag{4}$$

This process is executed for all the possible actions and the best action is selected as,

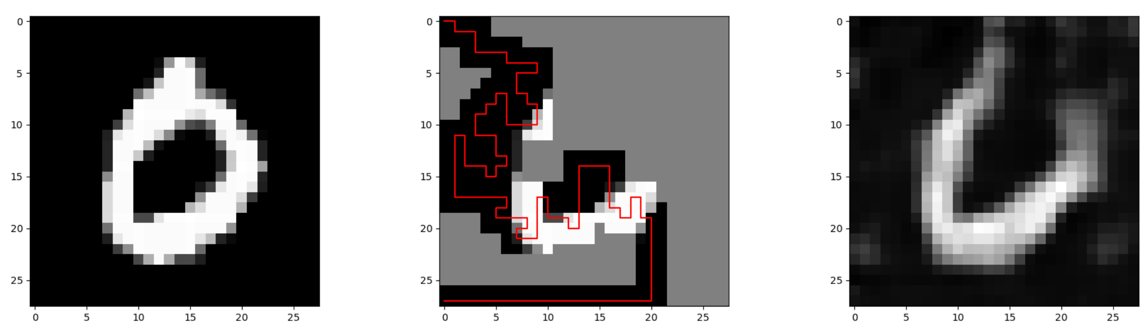
$$a^* = \arg\max_{a \in \mathcal{A}} (IG) \tag{5}$$

As shown in Algorithm 1, this greedy action selection is used until the probability of the estimated digit with estimated map will be higher than user-specified probability. Once it exceeds the threshold, the robot switch the action selector to GetShortestAction to reach the goal position with the minimum amount of path.

Figure 2 shows two example trajectories on digit 7 and 0. The average reward over 10 trials for digit 7 is 2, and for digit 0 is -34 if the p_{th} is set as 0.8. Note that there is no stochastic nature in the greedy solver so the obtained reward does not vary. As shown in Fig. 2 (c) and (f), the estimated world by the network is qualitatively similar to the ground truth map while the obtained trajectories aren't cost efficient. This is not surprising since the solver does not care the cost of path lengths during its exploration.



(a) (b) (c)



(d) (e) (f)

Figure 2: Examples of the ground truth ((a) and (d)), explored ((b) and (e)), and predicted ((c) and (f)) MNIST worlds. As shown in (a) and (d), the first world represents 7 and the next world represents 0. The red lines in (b) and (e) express the robot's trajectories by the greedy solver.

Step 2

i

Algorithm 3 shows the overview of the ϵ-greedy solver, which is the expansion of the greedy solver as implemented in Step 1. The disadvantage of the pure greedy solver is it would stuck into local optima. To give a chance of exploration, the ϵ-greedy algorithm balances the rate of taking random exploration and greedy exploration based on given ϵ value. If the uniform sampling of the value between 0 to 1 is lower than ϵ, it takes random action. Otherwise, it takes greedy action as shown in Algorithm 2.

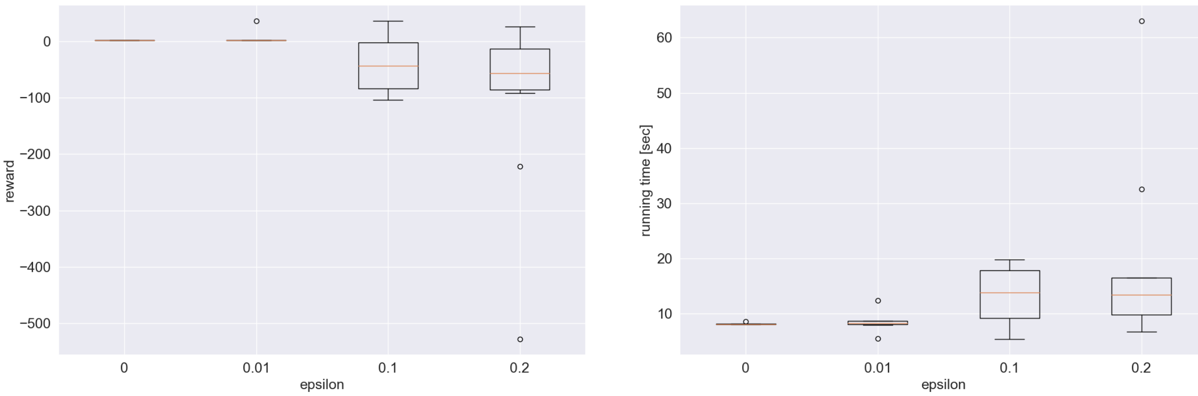
Algorithm 3 ϵ-greedy exploration

```
flag_greedy = true
while robot does not reach goal do
    if flag_greedy = true then
        if ε > UniformSampling() then
            a = GetRandomAction()
        else
            a = GetGreedyAction(map_exp)
        end if
    else
        a = GetShortestAction(goal)
    end if
    map_exp = UpdateMap(a, map_exp)
    map_est = WorldEstimationNetwork(map_exp)
    digit_est = DigitClassificationNetwork(map_est)
    if digit_est = digit then
        if p_digit_est(s, map_est) > p_th then
            flag_greedy = false
        end if
    end if
end while
```

The experiment is performed to explore MNIST's digit 7 world. The probability threshold p_{th} is set as 0.8, same as Step 1. ϵ values are changed from 0 to 0.2 and for every ϵ, 10 trials are executed. Table 1 shows average reward and running time and Figure 3 shows box plots of (a) reward and (b) running time with different ϵ value. Table 1 indicates that the best ϵ value is 0.01 in terms of maximizing average reward over 10 trials and it dramatically decrease as ϵ becomes larger. Figure 3 also indicates that ϵ negatively affects its exploration process in terms of average reward and computational efficiency if its value is 0.1 and 0.2.

Table 1: Average reward and running time

ϵ	0	0.01	0.1	0.2
average reward	2	5.4	-39.8	-104.6
average running time [sec]	8.18	8.47	13.26	19.10



(a) (b)

Figure 3: Box plots of (a) reward and (b) running time with different ϵ value. Note that the MNIST's digit is 7 and the probability threshold p_{th} is set as 0.8.

Discussion Questions

1

This informative path planning problem is formulated as a mixed integer problem as follows:

$$\text{minimize} \sum_{i=1}^{n} \sum_{j=1}^{n} \frac{c_{ij}}{f_{ij}} x_{ij}, \tag{6}$$

$$\text{subject to} \sum_{i=1} x_{ij} = 1, \tag{7}$$

$$\sum_{j=1} x_{ij} = 1, \tag{8}$$

$$x_{ij} \geq 0, \tag{9}$$

where c, f, and x represent cost function of trajectory, submodular information gain function, and variable that the robot visit a place or not as 1 or 0. The advantage of MIP is that it can consider the cost of searched trajectory for its optimization while the one I implemented only takes into account whether the path is information-rich one or not. On the other hand, the disadvantage of the MIP is that the performance will be arbitrarily bad due to integrality gap. If the gap between the solution of LP and ILP is huge, it might not be appropriate way to solve this problem using MIP.

2

The main motivation to implement the ϵ-greedy solver in Step 2 is to avoid getting stuck into local maxima and missing informative path by providing a chance to explore randomly while dismissing local maximal informative path. Another design criteria is to avoid giving computational burden for information gathering, since there are time limits to accomplish exploration. The algorithm I implemented can take action randomly depending on ϵ value, and as a result, the average reward of ϵ-greedy solver outperforms "pure" greedy solver, as shown in Table 1 and Fig. 3. Therefore, it can be said that the later one has the ability to seek more efficient path with its stochastic nature. As described in the course, branch and bound algorithm can find optimal solution by setting upper and lower bounds and systematically enumerating all the possible candidates. It will has better performance in terms of finding efficient path, but may requires more computational cost. Another algorithm is the one described in [1]. That algorithm purely seeks the best action based on network outputs, so may causes local optima. However, that also equips more sophisticated way for the robot's exploration: it uses three networks, one unbiased, and two biased networks to calculate information gain. Hence, the architecture can calculate better information gain against unexplored region so it is difficult to compare their performance in terms of avoiding local optima.

3

If the neural network provides a prediction of the correct digit with its uncertainty on the prediction, the greedy solver now needs to consider its stochastic nature for the action selection (policy). Since there is a possibility that the network incorrectly outputs the probability of classification, the solver will calculate expected information gain instead of actual one and select the action to maximize expected benefit.

4

In order to execute cooperative information gathering with multiple robots, the algorithm has to coordinate the robots' paths. Implicit coordination must be considered for ϵ-greedy solver to solve this problem efficiently. Suppose one robot plan paths within restricted depth by "imaging" world based on the network prediction. The local trajectory then is shared with other robots to avoid taking action towards the previously explored area. While other robots assume shared paths are fixed, it should be re-plan when it's necessary. This concept can be integrated by modifying one-step lookahead to multi-step lookahead and adding a re-plan function to the current implementation.

References

[1] J. A. Caley and G. A. Hollinger, "Environment prediction from sparse samples for robotic information gathering," *2020 IEEE International Conference on Robotics nad Automation*, Paris, France, 2020.