Exploration and Implementation of the MAXQ and Qlearning algorithms for the taxi problem [*]

MIHAI GROZA
Concordia University
mihai.groza@mail.concordia.ca

KHALED FOUDA
Concordia University
khaled.fouda@mail.concordia.ca

December 11, 2020

Abstract

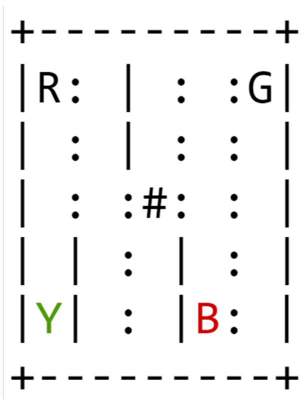
In this paper we re-define MAXQ and the taxi environment and Implement them in R. We then apply Qlearning to the same problem. Our conclusion is that MAXQ works as good as Qlearning for this problem. Our aim is illustrate the advantages of using hierarchical reinforcement learning methods. The implementations can be found in the project's github repository[4].

I. INTRODUCTION

Hierarchical reinforcement learning is a very interesting and promising topic. Here we explore the taxi environment as defined by OpenAi[2] with MAXQ[1] hierarchical method and Qlearning. In the second section we define the environment as we re-implemented it in R. In the third section we explore MAXQ method. First we define our model, the learning equations and important functions. We terminate the chapter by providing our results from applying the method to the taxi problem. In the fourth section we apply Q-learning to the taxi problem and observe the results. In the last section we draw our conclusions.

II. ENVIRONMENT

The taxi's goal is to first pick up the passenger from one of the four places (R,Y,G,B) where the current location of the passenger is shown in green. After picking up the customer, the next and final goal is to drive them to their destination which is one of the four places (R,Y,G,B) and the destination is shown in red.



```
+---------+
|R: | : :G|
| : | : : |
| : :#: : |
| | : | : |
|Y| : |B: |
+---------+
```

Figure 1: *generated by render()*

The state is defined as a set (taxi_row, taxi_column, passenger_location, destination_location) where:
taxi_row and taxi_column take values from 1 to 5 as we have 5x5 grid,
passenger_location and destination_location take values [1,2,3,4] for [R,Y,G,B]. Furthermore, if the passenger is in the taxi then they takes a location 5. That being said, we have $5 * 5 * 5 * 4 = 500$ different states.
The set of actions is (North, South, East, West, pick, drop) with ids 1 to 6.

The episode terminates only when the

[*]Reinforcement learning undergraduate class project

passenger is dropped at their destination and hence the agent receives a reward of +20. For any attempt to illegally picking or dropping the passenger, the agent receive a reward of -10 and exhibits no change in the state. Otherwise, they receive a reward of -1 for every move. Note that hitting the wall results in a reward of -1.

We have implemented the following functions for the environment:

- render(state) : prints out the environment at the current state.
- encode(s) : maps each state set to an integer between 1 and 500.
- decode(i): The inverse of encode(s).
- loc.indx(i): maps the four pick/drop locations (1,2,3,4) to a (row,column) pair.
- hitting.wallQ(a): returns true if the agent would hit the wall if they took the action a
- step(a): maps (s,a) pair into (r,s').

III. MAXQ method

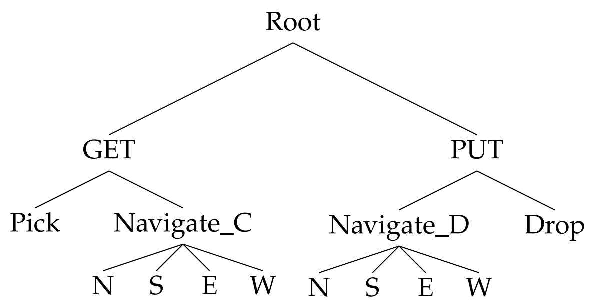
In our problem, the agent's goal is not only reaching the destination but is to also tp pick and drop the passenger at the right locations. From this we define the following sequence of subtasks needed to reach the goal.

Reach the customer $\Rightarrow$ Pick them up $\Rightarrow$ Drive to the destination $\Rightarrow$ drop them off

MAXQ algorithm is one of the hierarchical reinforcement learning methods which guarantees ultimate goal satisfaction only if a sequence of sub-goals were satisfied.
One the other hand, the algorithm's performance depends on how the structure of hierarchy is defined. MAXQ provides no way of learning that structure and it needs to be carefully designed by the researcher.
In MAXQ there are two types of actions: a primitive action which is the action that change the state of the environment and composite action which is a subtask. For our Implementations, we defined the following actions:

- Primitive actions:
  1. Move one step north (N)
  2. Move one step south (S)
  3. Move one step east (E)
  4. Move one step south (W)
  5. pick the passenger
  6. drop the passenger
- Composite actions:
  7. root: the main task.
  8. navigate to the passenger and pick them (GET)
  9. drive to the destination and drop the passenger (PUT)
  10. drive to the passenger's location (Navigate_C)
  11. drive to the destination (Navigate_D)

and Their hierarchical structure is defined as follows:



Where - starting at root - the agent has to recursively go down the graph satisfying each composite action. The root is satisfied if both get and put are satisfied and hence the episode ends. At each subtask i, the agent needs to repeatedly sample an action until they hit a terminal state and go back to the parent subtask. Therefore, a list of terminal states needs to be

defined for each subtask.
For example, at Navigate_C, the agent keeps sampling one of the four actions (N,S,E,W) until the agent reaches the passenger's location. Then, it reports back to the parent subtask "GET" and that subtask would either keep sampling "Pick" and "Naviagte_C", or go back to the parent "Root" if it hits one of it's terminal states. The only terminal state for root is when both "GET" and "PUT" reach their terminal states and the episode is over.

value function Estimates

Let (i, s) define the state of the environment at time t where i is the subtask running. Let also a be the action (sub-subtask) taken by subtask i at time t. Define

$$C^\pi(i,s,a) = (1-\alpha)C^\pi(i,s,a) + \alpha * \gamma^N * V[i,s']$$

$$V^\pi(i,s) = \begin{cases} (1-\alpha) * V^\pi(i,s) + \alpha * Reward(s'|s,i) \\ \text{if i is primitive} \\ max_{a\in A(i)} V^\pi(a,s) + C^\pi(i,s,a) \\ \text{if i is a subtask} \end{cases}$$

where α is the learning rate, γ is the discounting factor, N is the number of actions sampled by subtask i before hitting a terminal state, and $A(i)$ is the set of available actions to subtask i.
C^π is only defined for composite actions, i.e, for $i \in (7, 8, 9, 10)$ and it's similar to the usual action value function Q and it it takes into account the value function of the next state produced by a primitive action. Note that compsite actions never change the state s. The term γ^N is adapted from Semi-Markov decision model and it helps prioritizing the actions that leads to smaller number of steps.
The state value function for the primitive actions is a classic weighted average of rewards and for the composite actions is a maximization of a modified action value function. The state value function for the composite actions requires recursive calculations of the subsequent chosen actions.

Algorithm 1 Updating Value Function

```
0: procedure EVALUATEMAXNODE($i, s$)
     if i is a primitive action then
       do nothing
     else
       actions ← Action space of i
       for a in actions do
           EVALUATEMAXNODE($a, s$)
       end for
       $v(i,s) \leftarrow \max_{a\in actions} v(a,s) + c(i,s,a)$
     end if
       =0
```

Policy

The policy used is $\epsilon - greedy$. Only composite actions (subtasks) can sample actions and it's the action the maximizes the modified action value function above with probability $1-\epsilon$ and a random action with probability ϵ with the restriction of belonging to the action space -defined below- of that subtask.

Action Space

The action space for each subtask is the set of sub-nodes to that action.

```
Action.space <- function(.,i){
  # expects i to be from 7 to 13
  if(i==7){return(c(8,9))
  }else if(i==8){return(c(5,10))
  }else if(i==9){return(c(6,11))
  }else if(i==10 || i==11){return(1:4)}
}
```

Terminal States

Each subtask -subtree- runs recursively until it hits one of its terminal states. Terminal states do not necessarily represent the completion of that subtask, they can be states where the subtask is not needed at that state of the environment. We defined the following terminal states:

- When either the subtask Get or Navigate_C is active and the passenger is onboard.

- When either the subtask Put or Navigate_D is active and the passenger is not yet picked.
- When either of the navigation subtasks is active and the taxi has already arrived to the destination.
- When the episode is terminated.

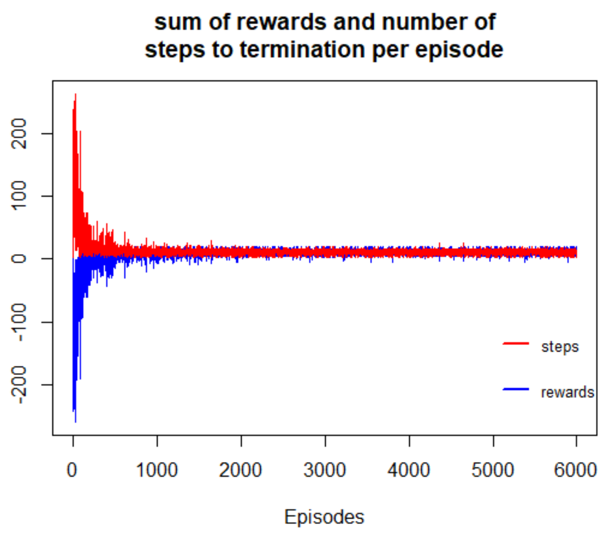
As for the primitive actions, they are always executed once and report back to their parent-node who then can make choice of whether to execute them again using the defined policy.

Learning Algorithm

Algorithm 2 Learning Function

```
0: procedure MAXQ0($i, s$)
  if i is primitive then
    map $(i, s)$ to $(i, s')$
    $V(i, s) \leftarrow (1 - \alpha) * V(i, s) + \alpha * r$
    $return(1)$
  else
    $count \leftarrow 0$
    while (i,s) not a terminal state do
      $a \leftarrow$ POLICY(i,s)
      $N \leftarrow$ MAXQ0(a,s)
      if episode is done then
        $return(count)$
      end if
      EVALUATEMAXNODE(i,s)
      $C(i, s, a) \leftarrow (1 - \alpha)C(i, s, a) + \alpha * \gamma^N * V[i, s']$
      $count \leftarrow count + N$
      $s \leftarrow s'$
    end while
  end if
end procedure=0
```

The learning function when called with an initial $i = 7$ it simulates a full episode from the beginning until the successful drop-off of the passenger. For larger scale problems and poorer design of the environment, the while loop might run forever before reaching a terminal state.



Results

MAXQ has 4 parameters to tune. The learning rate α, the discounting rate γ, the policy's ϵ and the number of simulated episodes.
After many trials, We chose the following parameters: $\alpha = .8, \gamma = .999, \epsilon = .01, n = 2000$. For α to be that high means we put more weight on newer episodes than old ones. Although in theory that should give worse results, in our experiment, α's best value is .8. Here below is a plot showing the evolution of rewards overtime and comparing the model with two different values of α. It is clear that the higher α achieved higher rewards.

Another note is the value of γ. The model performs well with gamma = 1 compared to values like .9 and .8. However, we chose the value .999 to filter out actions that requires very high number of steps. For example if the agent took 1000 steps to navigate to the destination, then $.999^{1000} = .367$ would give that action a lower value estimate.
Finally, we looked at the statistical summaries

min	median	mean	max	variance
-244	9	3	20	585.8

and from the first glance we notice the high variance. However, as in the plot above, for

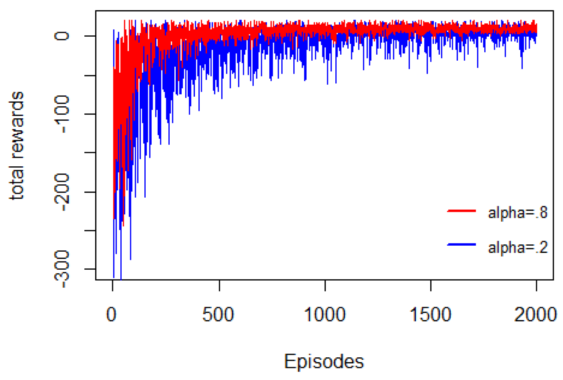


Figure 2: *evolution of rewards with two different learning rates*

the first 500 episodes the rewards fluctuate a lot and therefore, let us consider only the episodes starting from 500.

min	median	mean	max	variance
-17	10	9.3	20	30.7

Although the experiment still exhibits high variance, it is much lower than the first 500 episodes. Also the proximity between the mean and median shows some kind of stability in the rewards.

IV. Q-learning method

Q-learning uses the rewards given by the environment to learn about the best actions. More importantly, it does not require a model of the environment as opposed to other algorithms. It learns only through experience. It does so by updating a Q-table with the rewards experienced from the episodes. The values stored in the Q-table map to a state-action combination. These values represent the utility of each state-action. A better Q-value implies a higher reward if the action is taken.

Value Function

To begin, the Q-table is initialized using arbitrary values. Then, as the agent goes through more episodes, the table will be updated using the following formula:

$$Q(s,a) = Q(s,a) + \alpha(r + \gamma Q(s',a') - Q(s,a))$$

α: The learning rate is a parameter that determines the step size of each Q-value update. It takes a value between 0 and 1.

γ: The discount factor determines how much weight you wish to give future rewards. A low gamma will make our agent value immediate rewards instead of long-term ones. Gamma takes a value between 0 and 1.

This formula takes the reward received for the current state-action pair and sums it to the difference in Q-values of the current state-action pair and the future optimal one. If the future optimal state-action pair is higher than the current state-action, the current Q-value will increase. Otherwise, it will decrease. Additionally, you can tune the parameters to achieve optimal learning.

State Space and Actions

We have a 5x5 2d environment. This gives us 25 possible locations. Moreover, there is 4 destinations and 5 passenger locations. In total, our environment will have 500 states (5x5x5x4).

Our agent will also have the usual 6 actions. This consists of south, north, east, west, pickup and dropoff.

Therefore, the Q-table for this environment will be size 6x500.

Policy

For this problem, we will use an epsilon-greedy policy. This will act as an exploration mechanism. For each action, at epsilon proportion of time, the agent will pick a "non-optimal" action in order to discover more rewarding state-action pairs. In this case, using a high epsilon will allow the agent to learn the optimal Q-table very quickly, but will underperform a lower epsilon policy in the long run.

Algorithm

Algorithm 3 Learning Function

```
0: procedure Q-LEARNING
  Initialize Q-table
  while S′ is not terminal state do
    Start the episode
    Choose action a in state s using $\epsilon$ greedy policy
    Perform action into state S′ and receive reward r
    Update Q-value
  end while
  end procedure=0
```

Results

The training was done using an epsilon of 0.017, a gamma of 0.99 and a learning rate of 0.4.

We used a high gamma because most rewards are negative and it takes the agent many steps until it reaches the positive reward. Therefore, the agent must prioritize long term rewards.

A too high learning rate will make the learning jump over minima but a too low learning rate will take too long to converge. Therefore, we chose one in the middle such as 0.4.

A low epsilon will eventually outperform a high epsilon policy due to it exploring more. We opted for training using an epsilon greedy method but testing using a 0 epsilon policy.

The figure shows the convergence of the algorithm.

The Q-table was initialized at 0. Therefore, it results in very high variance in terms of rewards as the agent is still learning the state space.

We can also see that the algorithm converges very quickly. In this case, it takes less than 500 episodes with an epsilon of 0.017.

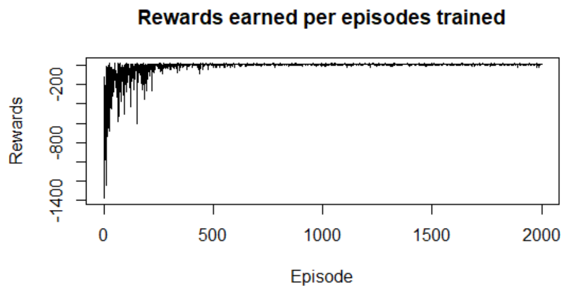


Figure 3: *evolution of rewards with an epsilon of 0.017*

min	median	mean	max
-1296	9	-8.7335	20

These statistics are based on the rewards of the 2000 episodes.

V. Final thoughts

Although we had satisfying results using the MAXQ method. It took noticeably more time to learn than other methods. However, it converged well optimally.
One the other hand, one disadvantage to using the rewards to attest the performance is that there is no single optimal reward. The optimal reward is different for different initial state and hence rewards will not converge to a single value. However, by analyzing the range and mean of the results we can safely attest the performance.
The taxi problem might not be the optimal test for MAXQ's performance. MAXQ might shine in more complicated problems where the usual methods are unable to follow the sequence of subtasks. For future project, we will attempt to apply the method to a more complicated problem.

In the case of the Q-learning algorithm, it is a simple one. Therefore, it is the best approach for MDPs that have a small number of states such as the taxi problem. It is very easy to implement and converges very quickly. The disadvantage that it does not scale very well with more complex problems that cannot be fit on a Q-table.

References

[1] Thomas G. Dietterich *Hierarchical Reinforcement Learning with the MAXQ Value Function* [*https://arxiv.org/abs/cs/9905014*].

[2] OpenAi *Taxi environment defined by OpenAi* [*https://gym.openai.com/envs/Taxi-v3/*]

[3] Andrew Barto and Richard S. Sutton *Reinforcement Learning: An Introduction*

[4] Khaled Fouda and Mihai Groza *Implementation of MAXQ and taxi environment* [*https://github.com/khaledfouda/taxi-env-with-MAXQ-and-Qlearning-in-R*]