Reinforcement Learning: Exploring UCBVI

Mathias Vigouroux

École Normale Supérieure, Master MVA
mathias.vigouroux@ens-psl.eu

April 3rd 2024

Figure 1: Painting coming from prompting Dall-E with the abstract.

Abstract

This project is related to the class of Reinforcement Learning [1], for the two last sessions given by E. KAUFMANN. It explores conducts both an analytical and empirical study on the Upper Confidence Bound Value Iteration (UCBVI) algorithm within a discretized Mountain Car environment, juxtaposing it with the model-free Q-learning. Especially, it showcases UCBVI's proficiency in harmonizing exploration with exploitation for enhanced policy learning. Moreover, it also study the effects of exploration bonuses on performance.For reprodcutory purpose, the code used for this report can be found in the following Google Colab notebook [2].

[1] Reinforcement Learn ing https://www.master-mva.com/cours/reinforcement-learning-2/
[2] Code: https://colab.research.google.com/drive/1uPmy63poU_Z3X1KIsD8HG2E_2rnqCSxF?usp=sharing

1 Exploration-Exploitation in Reinforcement Learning

In this homework, you will implement the **UCBVI** algorithm, for exploration in MDPs with finite states and actions and a **finite horizon** criterion. In a finite horizon criterion, the value function of a policy (starting from the h step of the episode) is

$$V_h^\pi(s) = \mathbb{E}^\pi \left[\sum_{\ell=h}^{H} \gamma^{\ell-h} r_\ell \middle| s_h = s \right]$$

where the discount parameter $\gamma \in (0, 1]$ is often set to $\gamma = 1$.

1.1 Environment

Our goal is to learn a good policy in a Mountain Car [3]. The Mountain Car environment as implemented in gymnasium has a continuous state space. The environment consists of a car placed stochastically at the bottom of a sinusoidal valley, with the only possible actions being the accelerations that can be applied to the car in either direction. The goal of the MDP is to strategically accelerate the car to reach the goal state on top of the right hill.

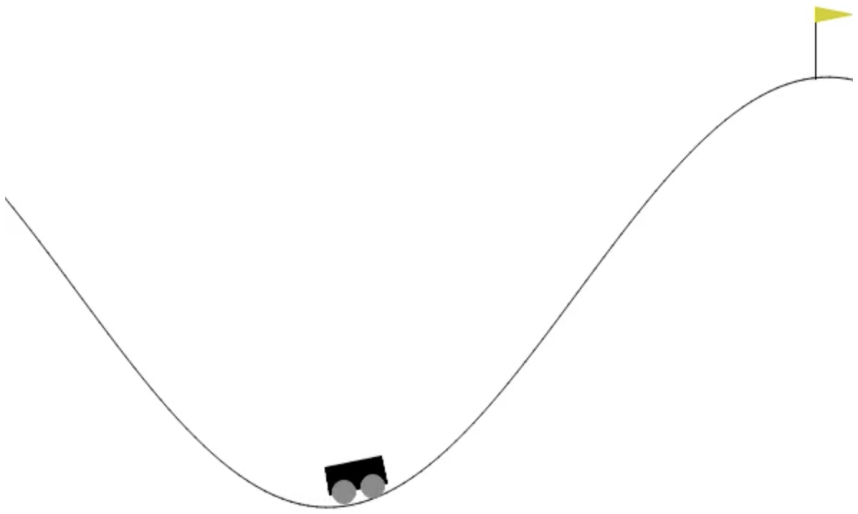


Figure 2: Screenshot of the Mountain Car environment

In order to, apply UCBVI see 3.1, we will use a discretized version of the environment, from the rlberry library.

From this, it is possible to create a reward wrapper, for which, the agent gets a reward 0 in states that are not the top of the right hill. Once the goal state is reached, the agent stays in this state and gets a reward 1 until the end of the episode. This is therefore not just a measure of being able to climb the hill but also how fast the agent was able to climb the hill.

2 Implementation of backward induction

In a finite-horizon MDP, the optimal Bellman equations given a recursion that can be used to compute the optimal value function. We have $V_{H+1}^\star(s) = 0$ for all s and for $h \leq H$,

[3] Mountain Car: https://gymnasium.farama.org/environments/classic_control/mountain_car/

$$Q_h^\star(s,a) = r(s,a) + \gamma \sum_{s\prime \in \mathcal{S}} p(s\prime|s,a) V_{h+1}^\star(s\prime) \quad \text{and} \quad V_h^\star(s) = \max_{a \in \mathcal{A}} Q_h^\star(s,a).$$

Moreover, the optimal policy is deterministic but **non-stationary** and satisfies $\pi_h^\star(s) = \text{argmax}_a Q_h^\star(s,a)$.

The following algorithm enables us to code it :

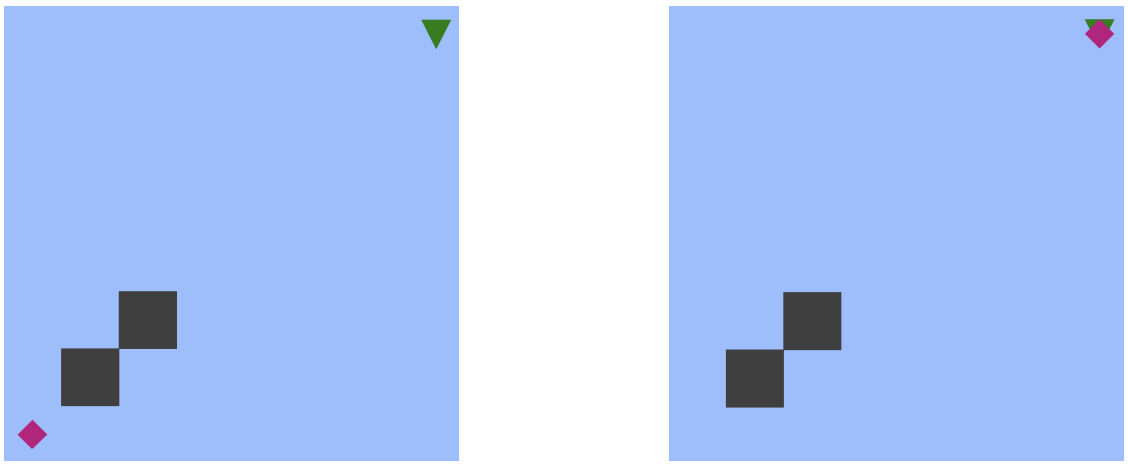
Algorithm 1 Backward Induction for Optimal Q-function

Require: P: Transition function matrix with dimensions (S, A, S)
Require: R: Reward function matrix with dimensions (S, A)
Require: H: Planning horizon
Require: γ: Discount factor, typically set to 1.0 for finite-horizon problems
Ensure: Q: Optimal Q-function with dimensions $(\text{horizon}, S, A)$

$S, A \leftarrow$ dimensions of P ▷ Number of states and actions
Initialize V to a zero matrix of dimensions $(H+1, S)$
Initialize Q to a zero matrix of dimensions $(H+1, S, A)$
for $h = H-1$ to 0 (inclusive) in reverse order **do**
    for each state s from 0 to $S-1$ **do**
        for each action a from 0 to $A-1$ **do**
            $Q[h,s,a] \leftarrow R[s,a] + \gamma \times \sum(\text{P}[s,a,:] \times V[h+1,:])$
        end for
        $V[h,s] \leftarrow \max(Q[h,s,:])$ ▷ Optimal value for state s at horizon h
        if $V[h,s] > H-h$ **then**
            $V[h,s] \leftarrow H-h$ ▷ Clip $V[h,s]$ if necessary
        end if
    end for
end for
return Q

Yet, one cannot try this function in the mountain car environment, as the expected rewards and transition probabilities cannot be easily computed, and are not embedded in the environment.

However, it is possible to check that the algorithm is functioning correctly on a simpler Gridworld environment 3, as P and R are properties of the environment.



(a) Start of the task

(b) End of the task

Figure 3: Using the Value Iteration in the GridWorld: the little red square needs to reach the green triangle while avoiding the big black squares.

3 Implementation of UCBVI

3.1 Presentation of the algorithm

Introduced in 2017, in the work of Azar et al. [2], the algorithm of Upper Confidence Bound Value Iteration (UCBVI) works as follows:

* In each episode t, the agent has observed n_t transitions $(s_i, a_i, r_i, s_{i+1})_{i=1}^{n_t}$ of states, actions, rewards and next states. * We estimate a model of the MDP as:

$$\textbf{rewards}: \quad \widehat{R}_{t(s,a)} = \frac{1}{N_t(s,a)} \sum_{i=1}^{n_t} \mathbb{1}\{s = s_i, a = a_i\} r_i \textbf{transitions}: \quad \widehat{P}_t(s\prime|s,a) = \frac{1}{N_t(s,a)} \sum_{i=1}^{n_t} \mathbb{1}\{s = s_i, a = a_i,$$

where

$$N_t(s,a) = \max\left(1, \sum_{i=1}^{n_t} \mathbb{1}\{s = s_i, a = a_i\}\right)$$

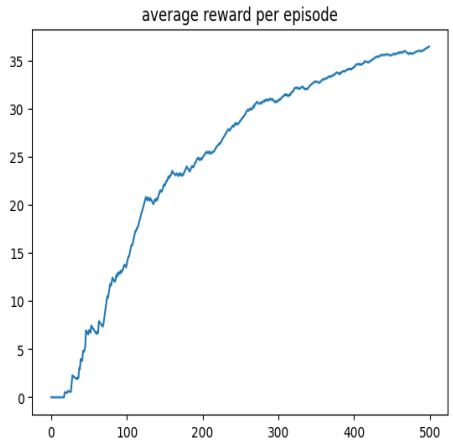
* We define an appropriate exploration bonus, which is some function of the number of visits, for example

$$B_t(s,a) \propto \sqrt{\frac{1}{N_t(s,a)}}.$$

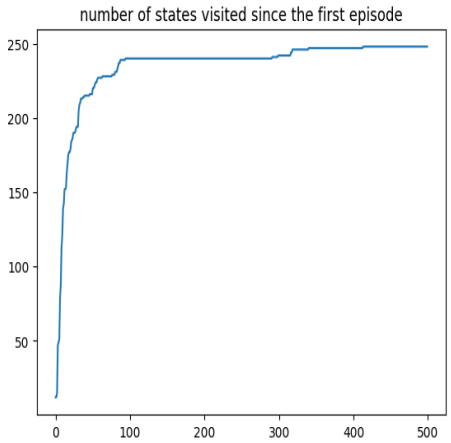
For this first experimentation, it is actually exactly equal to it. * Then, in episode t, we compute $\{Q_h^t(s,a)\}_{h=1}^H$ as the (H-horizon) optimal value functions in the MDP whose transitions are $\widehat{P}_t$ and whose rewards are $(\widehat{R}_t + B_t)$. At step h of episode t, the agent chooses the action $a_h^t \in \arg\max_a Q_h^t(s,a)$.

3.2 Experimentation

To check whether the algorithm is working, it is possible to visualize the amount of rewards gathered in episodes with the default bonus, as well as the number of visited states since the beginning (which measures how well the environment is being explored).



(a) Evolution of the reward



(b) States explored

Figure 4: Using this first Bonus, the agent is able to complete the task and even progress and go faster to reach the goal

A little struggle Given the initial plot in solution (which is below) I struggled for a long time because I tried to reach a similar curve. However, I was between 0 and 1. It took me time

before realizing that I was learning something, that it was just because I had put a condition = done that I stopped my algorithm earlier.

However, we can see that the algorithm started to reach a plateau of performances around 300 iterations. It can be seen that after 20 trials the agent is able to reach the the goal. However, this task is quite challenging since it is necessary to start by going further from the goal to take some momentum before reaching it.Thus, I would be tempted to say that the algorithm starts to behave well around 20 iteratons.

Displaying the value of this policy is not possible, as it would require to perform policy evaluation, and as P and R are unknown, Monte-Carlo evaluation is the only option. Let's keep this for later. For now a proxy is to visualize $\max_a Q(s,a)$ when π is greedy wrt to Q. For the optimistic policy, Q is the optimistic Q function.

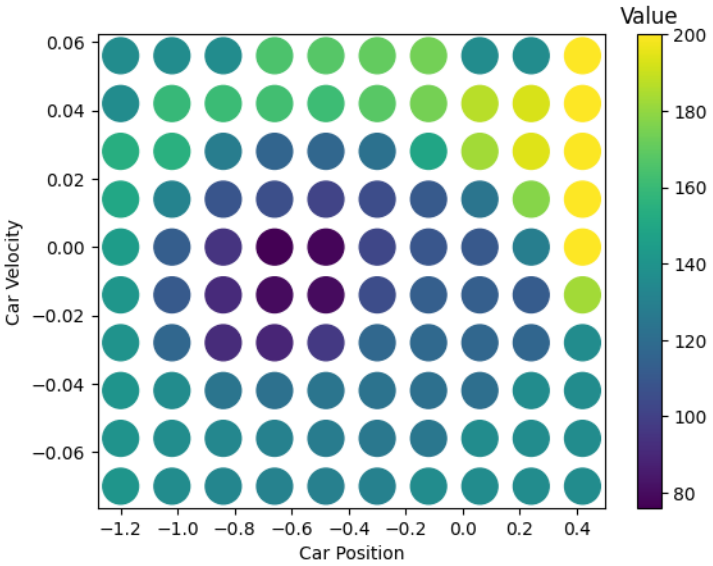


Figure 5: Display the Q-states

3.3 Bounding the regret

In the context of reinforcement learning, regret R_T measures the difference in value between the optimal policy and the policy chosen by the algorithm over T episodes. The regret after T episodes is the sum of differences in expected rewards between the optimal policy and the chosen policy at each episode, cumulated over T episodes. Namely the regret is given by :

$$R_T = \sum_{t=1}^{T} V^\star - V^{\pi_t}$$

In theory, any algorithm with an upper bound $\mathbb{E}[R_T] \leq B_T$ on its expected regret can be used to output an ε-optimal policy with high probability using the following trick: run the algorithm for a sufficiently large T and output a policy $\widehat{\pi}_T$ chosen at random among the first T policies used by the algorithm. The point of this question is to quantify how long.

Therefore, for a given T, it is possible to compute an upper bound on the probability $\mathbb{P}(V^\star - V^{\widehat{\pi}_T} > \varepsilon)$ that depends on T, ε and B_T. First, using the formula of the total probabilities, the expression yields :

$$\mathbb{P}(V^\star - V^{\widehat{\pi}_T} > \varepsilon) = \sum_{t=1}^{T} \mathbb{P}(V^\star - V^{\widehat{\pi}_T} > \varepsilon \mid \widehat{\pi}_T = \pi_t)\mathbb{P}(\widehat{\pi}_T = \pi_t)$$

.
Then, as we select $\widehat{\pi}_T$ uniformly among the first T policies, then : $\mathbb{P}(\widehat{\pi}_T = \pi_t) = \frac{1}{T} \forall t = 1, ..., T$. This yields :

$$\mathbb{P}(V^\star - V^{\widehat{\pi}_T} > \varepsilon) = \frac{1}{T} \sum_{t=1}^{T} \mathbb{P}(V^\star - V^{\pi_t} > \varepsilon)$$

Recall: Markov's Inequality
Given a non-negative random variable X and a positive value a, Markov's inequality is:

$$\mathbb{P}(X \geq a) \leq \frac{\mathbb{E}[X]}{a}$$

Applying to the regret yields :

$$\frac{1}{T} \sum_{t=1}^{T} \mathbb{P}(V^\star - V^{\pi_t} > \varepsilon) \leq \frac{1}{T} \sum_{t=1}^{T} \frac{\mathbb{E}[V^\star - V^{\pi_t}]}{\varepsilon}$$

Thus, by the linearity of the expectancy and definition of the regret, it yields :

$$\frac{1}{T} \sum_{t=1}^{T} \frac{\mathbb{E}[V^\star - V^{\pi_t}]}{\varepsilon} = \frac{\mathbb{E}[R_T]}{T\varepsilon}$$

Using the bound on the expected regret yields the **final results** :

$$\mathbb{P}(V^\star - V^{\widehat{\pi}_T} > \varepsilon) \leq \frac{B_T}{T\varepsilon}$$

Deduce the order of magnitude of the number of episodes needed by UCB-VI to output a ε-optimal policy with probability $1 - \delta$. Indeed, to guarantee that a policy $\hat{\pi}_T$ is ϵ-optimal with probability $1 - \delta$, we set the upper bound equal to δ and solve for T:

$$\delta = \frac{B_T}{T\epsilon} \implies T = \frac{B_T}{\delta\epsilon}$$

This analysis highlights the trade-off between computational cost (number of episodes) and the quality plus confidence of the policy obtained. Indeed, the result shows that the number of episodes needed to ensure an ϵ-optimal policy with high probability scales inversely with both ϵ and δ. As we desire more precision (ϵ gets smaller) or higher confidence (δ gets smaller), the number of required episodes increases.

3.4 Trying different bonuses

Looking around the internet, I was able to find different forms of bonuses.

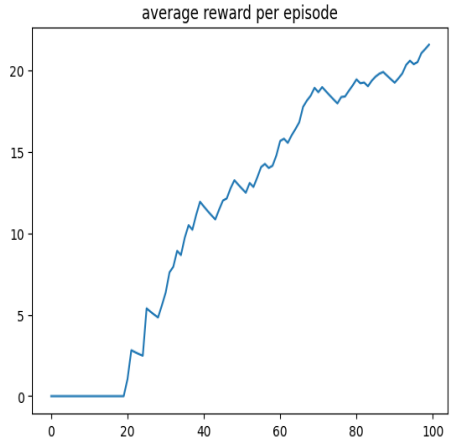
In the manner of the bonus used previously, all bonuses founded used the square root of $N_t(s, a)$. However, some of them also added the logarithm of the time step in the numerator. This was already present in the original paper of UCBVI [2].

However, I stub upon an interesting blog post on Stack discussing an issue in using different types of bonuses by changing factors inside of it [4].

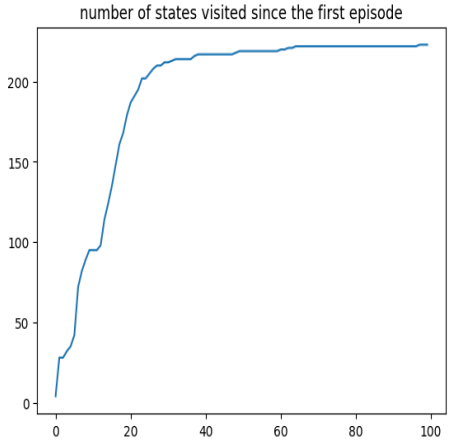
[4] Bonuses open questions: https://ai.stackexchange.com/questions/26187/ucb-like-algorithms-how-do-you-compute-the-exploration-bonus

The main question arrived from the fact that in [3] they were using a factor 1.5 inside the bonus yielding, $B_t(s,a) = \sqrt{\frac{2 \ln t}{N_t(s,a)}}$ and there were no clear reasons why, since in the historical literature it was a value of 2 [1].

Not being able to find a clear analytical explanation, I decided to test it empirically.

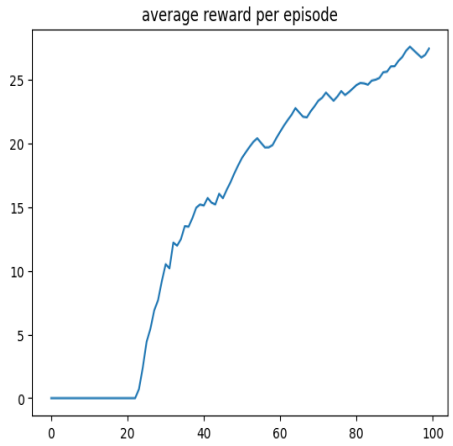


(a) Evolution of the reward

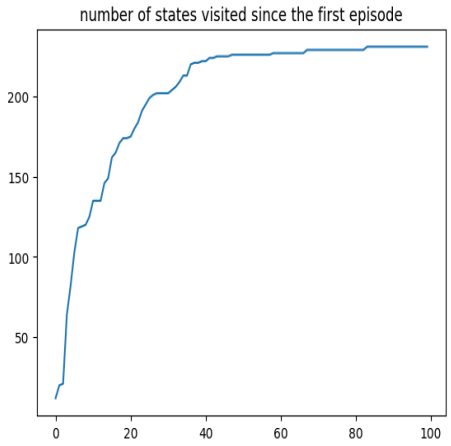


(b) States explored

Figure 6: An intermediary exploration Bonus, $B_t(s,a) = \sqrt{\frac{1.5 \ln t}{N_t(s,a)}}$ inspired from [3]



(a) Evolution of the reward



(b) States explored

Figure 7: A large exploration Bonus, $B_t(s,a) = \sqrt{\frac{2 \ln t}{N_t(s,a)}}$ inspired from [1]

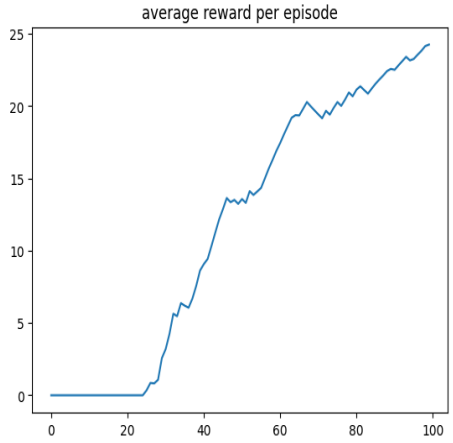
At first sight, it thus seemed that the values of 2 was better than 1.5 and I rushed toward a fast explanation : the task demands a lot of exploration and therefore a larger exploratory bonus is better.

Yet, this first intuition was broken by the fact that I once use a bonus with a constant of 1 then it yiedls better than with 1.5.

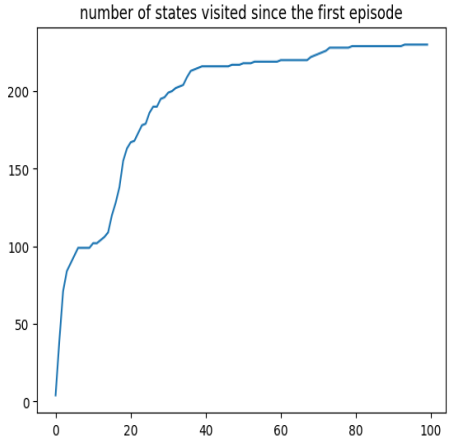
In a nutshell, $\mathcal{B}(2) > \mathcal{B}(1) > \mathcal{B}(1.5)$, where $\mathcal{B}(\bullet) = \sqrt{\frac{\bullet \times ln(t+1)}{N_t(s,a)}}$

3.5 Discounted MDP

Our goal was to find a policy that manages to get the car up the hill. We chose to model this as solving an episodic MDP with a large enough horizon H. In other words, the agent will do 200 interactions with the environment before the game is stopped



(a) Evolution of the reward



(b) States explored

Figure 8: A large exploration Bonus, $B_t(s,a) = \sqrt{\frac{\ln t}{N_t(s,a)}}$

The remaining question is to know if this task could have been modelled solving a discounted MDP. In a discounted MDP, there isn't such an abrupt terminaison. The gamma discounted criterion in reinforcement learning and decision-making models uses a discount factor, γ, to value future rewards less than immediate rewards. Thus, the value of a sequence of rewards $R_0, R_1, R_2, \ldots$ received over time, under this criterion, is calculated using the following equation:

$$V = R_0 + \gamma R_1 + \gamma^2 R_2 + \gamma^3 R_3 + \cdots = \sum_{t=0}^{\infty} \gamma^t R_t$$

With γ being the discount factor, verifying $(0 \leq \gamma < 1)$.

Thus, this procedure can be seen as an agent who have a probability of stopping to receive a reward at each time step of $1 - \gamma$.

In the end, the termination procedure thus follows a geometric law, $\mathcal{G}(1-\gamma)$. Thus, it has an expectancy of $\frac{1}{1-\gamma}$.

As a consequence, to have a horizon of H, it is necessary to have:

$$H = \frac{1}{1-\gamma}$$

.

Numerical application: During this homework we always used $H = 200$, this yields:

$$\gamma = 1 - 1\frac{1}{200} = 0.995$$

3.6 Model Free alternative

One possibility is then to compare the best model free algorithm found previously, to a model free alternative.

Naturally, I tended toward a Q-learning algorithm. TThe idea is to try to make some variant of the (discounted) Q-Learning algorithm work for this problem. Learning will however still proceed in episodes of length H, with a reset, and we will similarly monitor the (undiscounted) amount of rewards gathered in each episode, for comparison with UCBVI.

3.6.1 Starting ε-greedy exploration

You may explore one of the following possibilities: combining ε-greedy exploration with a careful initialization of the Q-value.

Algorithm 2 Discounted Q-Learning with ε-Greedy Exploration

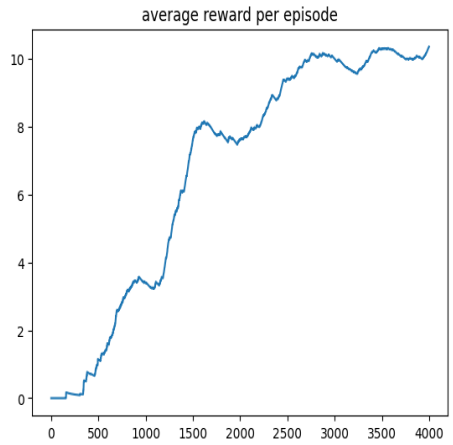
Initialization: Initialize Q-values optimistically to encourage exploration. For problems with sparse rewards, such as Mountain Car, initialize Q-values slightly above the expected rewards to promote exploration.
for each episode **do**
    Reset the environment
    for each step of the episode **do**
        With probability ε, choose a random action a (explore)
        Otherwise, choose action a with the highest Q-value for the current state s (exploit)
        Execute action a, observe reward r and next state s'
        Update Q-value for the state-action pair (s, a) using:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

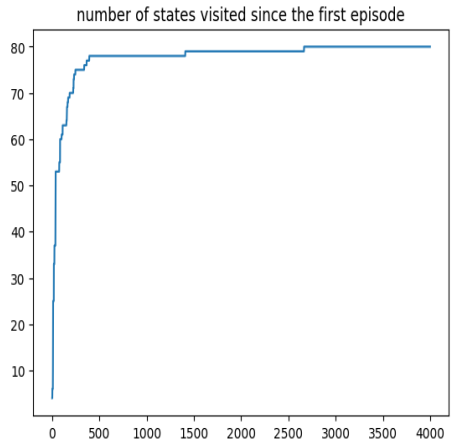
▷ α is the learning rate, γ is the discount factor

    end for
    Monitor the total undiscounted rewards gathered in the episode
end for

This is depicted in 9



(a) Rewards for a greedy Q-learning



(b) States explored for a greedy Q-learning

Figure 9: The agent has less good performance with just a greedy Q-learning algorithm

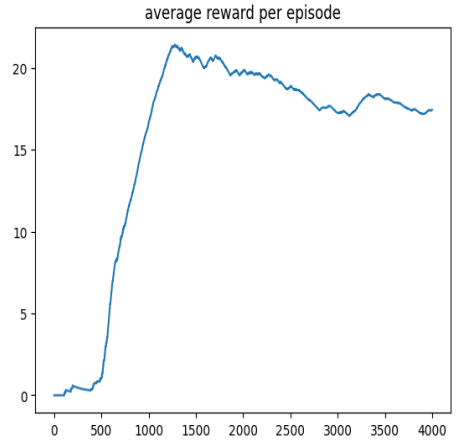
What is important to notice is that it is much quicker to compute.

3.6.2 Giving bonuses all the time

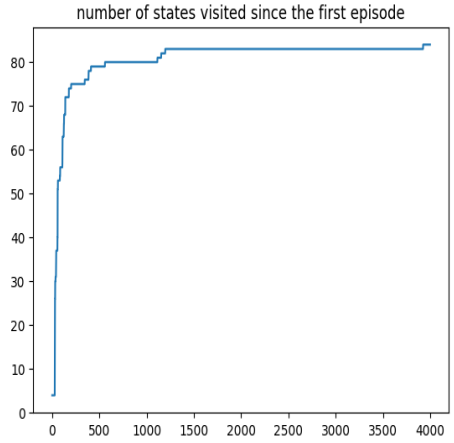
One suggestion in the homework was then to replace the ε-Greedy Exploration with a bonus at every time. Doing so gave interesting results as it can be in the following Figure 10.

3.6.3 Creating ε -random and bonuses all the time agent

My idea was then to combine the ε-greedy agent with the constant bonus agent. Therefore, the agent was always receiving a bonus excpetd ε-times when it would be random. This led to the following results in Figure 11.

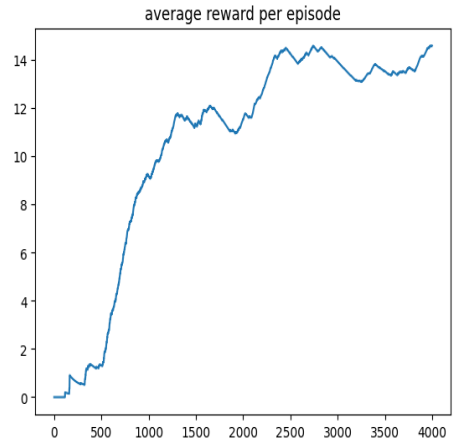


(a) Rewards for a bonus Q-learning

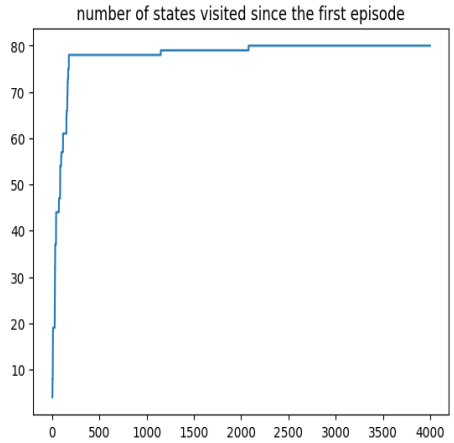


(b) States explored for a bonus Q-learning

Figure 10: The agent with bonus all the times is better than the ε-greedy one but then has its performance decreasing



(a) Reward for an exploration governed both by randomness and bonuses Q-learning



(b) States explored for an exploration governed both by randomness and bonuses Q-learning

Figure 11: This was not an improvement compared to constant bonuses

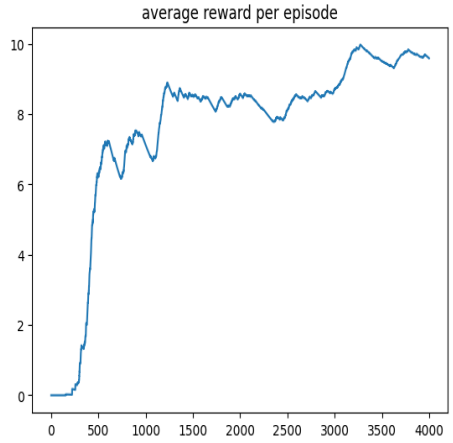
3.6.4 ε-bonuses and then greedy agent

The last attempt was to create an agent that was exploring, following a bonus based exploration ε-often.

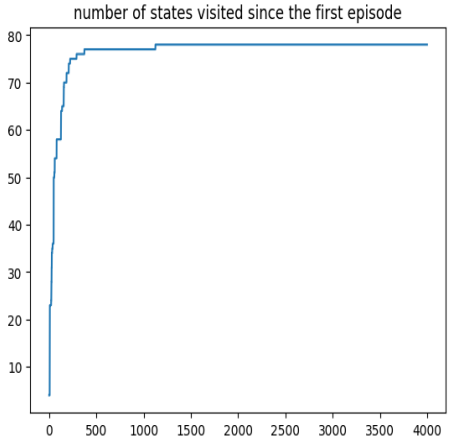
This led to the following result, which was not better 12 :

4 Complexity comparison

Q-learning algorithms are much quicker compared to the model base algorithm for one iteration, but they might need more samples to learn from, and this is exactly what I did in the previous experimentation comparing experimentation running for a 100 or at least 500 iterations for the model base, UCBVI algorithm, and the Q-learning algorithm which lasted for 5000 iterations. Thus, there is one remaining question, which is to really assess the complexity difference between these two families of models. The goal is therefore to propose an experimentation to compare how many samples (i.e., interactions with the environment) each algorithm requires to achieve a certain level of performance.



(a) Reward for an exploration governed both by ε-bonuses Q-learning



(b) States explored for an exploration governed both by ε-bonuses Q-learning

Figure 12: This was not an even worse than before even with a large $\varepsilon = 0.1$ compared to constant bonuses

To begin with, it is necessary to establish a clear performance criterion. This could be reaching a predefined reward threshold, achieving a reward within a certain percentage of the optimal policy's reward, or demonstrating a certain level of proficiency in a task (e.g., solving a maze, winning a game). The performance criterion should be consistent across comparisons.

Then, it would be necessary to define a proper Environment, measure the sample complexity with the performance criterion given above and then repeat the experiment with different hyper-parameters, or at least random seeds. Once this is done, the goal will be to do statistical measures to asses really if one is better than the other.

5 Conclusion

To conclude, this report provides a, analysis of the Upper Confidence Bound Value Iteration (UCBVI) algorithm applied to a discretized Mountain Car environment, contrasting it with the model-free Q-learning approach to highlight the nuances and trade-offs in learning efficiency and sample complexity. Through experimentation, the study demonstrates UCBVI's ability to adeptly balance exploration and exploitation for optimal policy learning. It then, discuss on the impact of exploration bonuses and the comparative effectiveness of model-based versus model-free methods.

This shows the sensibility to parameters in an RL task especially when the rewards are sparse.

References

[1] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2):235–256, 2002.

[2] Mohammad Gheshlaghi Azar, Ian Osband, and Rémi Munos. Minimax regret bounds for reinforcement learning. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 263–272. PMLR, 06–11 Aug 2017.

[3] Branislav Kveton, Zheng Wen, Azin Ashkan, and Csaba Szepesvari. Tight Regret Bounds for Stochastic Combinatorial Semi-Bandits. In Guy Lebanon and S. V. N. Vishwanathan, editors, *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, volume 38 of *Proceedings of Machine Learning Research*, pages 535–543, San Diego, California, USA, 09–12 May 2015. PMLR.