Research Internship Report

Cooperative Reinforcement Learning in Multi-Agent Scenarios

Paul Caron

August 2019

Abstract

Reinforcement learning has received great interest in the last decades as it succeeded in reaching and even surpassing human performance in various fields, especially games like Go. In addition to these great breakthroughs, reinforcement learning is becoming very popular in the development of cutting-edge sector requiring autonomous behaviors from machines like robotics and self-driving cars. Among the various fields of RL, cooperative multi-agent RL, the area that puts together autonomous agent in order to achieve collective goals, is gaining momentum. To the usual challenges of RL like *exploration-exploitation*, MARL adds new challenges like coordination that makes the learning task way harder.

After redefining key elements of SARL and MARL, the objective of this report is to explain the progression in solving MAMDPs, from basic MARL algorithms and sublinear-regret SARL algorithms to the problem of doing near-optimal learning in multi-agent scenarios. After mentioning the main challenges of near-optimal learning in MARL and the various approaches considered, we introduce the algorithm that we designed, *Pessimistic Q-learning with exploration policies* that aims at computing the max-Q functions. This algorithm is then tested in a challenging environment.

Summary

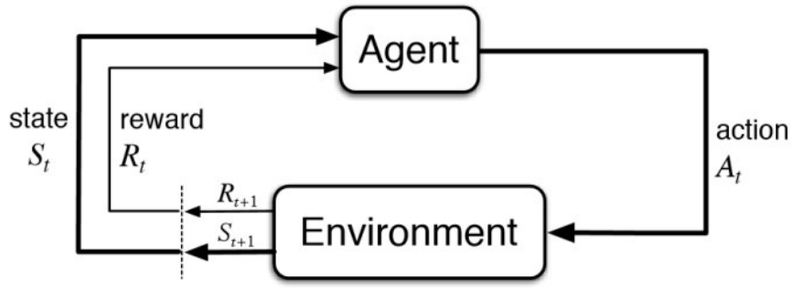

Figure 1: The formalism of Reinforcement Learning

1 Reinforcement learning overview

1.1 What is reinforcement learning?

Along with supervised learning and unsupervised learning, reinforcement learning is one of the three main branches of machine learning. In Reinforcement learning, we are concerned with *agents* taking *actions* in an unknown *environment* in order to maximize a *cumulative reward*. In each *state*, an agent is supposed to choose an *action* that implies a transition to another *state* and gives the agent an immediate *reward*. Thus, the *learning task* is made by *interacting* with the environment. Reinforcement learning is studied in many fields and has various applications, in scientific fields such as robotics and logistics, or in more applied fields such as online bidding and recommendations (cf. bandits).

1.2 Reinforcement learning's formal framework

In this section, let us denote $\mathcal{S}$ the set of states, $\mathcal{A}$ the set of actions (alternatively it can be assumed that for any $s \in \mathcal{S}$, it exists a set of actions $\mathcal{A}_s$). At time t, the next state and the transition reward are randomly sampled according to a distribution depending on the history $h_t = (s_t, a_t, ..., s_1, a_1)$: $P_{h_t}(\cdot)$ denotes the distribution probability over transitions after history h_t ($P_{h_t}(\cdot) = \mathbb{P}(\cdot \mid h_t)$) and $R_{h_t}(\cdot)$ is the random variable associated to the reward after history h_t.

1.2.1 Markov Decision Process

In reinforcement learning, the environment is commonly modeled as a Markov Decision Process (MDP): $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, r, \gamma \rangle$. $\mathcal{S}$ and $\mathcal{A}$ respectively denote the finite state and action sets. The Markovian assumption allows us to consider markovian transition dynamics: $P(s_{t+1}|s_0, a_0, \ldots, s_t, a_t) = P(s_{t+1}|s_t, a_t)$. The reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is defined as $r(s, a) = \mathbb{E}[R_{(s,a)}]$. Finally, $\gamma \in [0, 1]$ is the discout factor such that the objective is to maximize the cumulative reward (or return) at each step t: $G_t = \sum_{t'=t}^{+\infty} \gamma^{t'-t} R_{t'}$.

1.2.2 Policy

In Reinforcement Learning, the objective of an agent is to learn an optimal behavior in the environment. This behavior is represented by a *policy*. A *policy* is a mapping from states to actions such that an agent sticking to a given *policy* knows how to choose its action in any state. Formally, a (deterministic) *policy* is a function $\pi : \mathcal{S} \to \mathcal{A}$ such that, in state s, an agent chooses action $\pi(s)$. More generally, a policy π can also be stochastic. In this case, in state s, the agent chooses action a with probability $\pi(a|s)$. Then, for any state s, $\pi(\cdot|s)$ is a probability distribution over $\mathcal{A}$.

1.2.3 Value function and Bellman operator

Given the definition of a policy, it is necessary to define the *value* of a policy in order to compare them and define an *optimal* policy. Thus, for any policy π, we can define the value function V by:

$$V^\pi(s) = \mathbb{E}_\pi[G_t|s_t = s] \tag{1}$$

where $\mathbb{E}_\pi$ is the expected-value under the assumption that the agent follows policy π at each time step.

A policy π^* is said to be an *optimal policy* if, for any policy π and state $s \in \mathcal{S}$, $V^{\pi^*}(s) \geq V^\pi(s)$.

Now, given a policy π, let us also define $P^\pi \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{S}|}$ and $R^\pi \in \mathbb{R}^{|\mathcal{S}|}$ by:

$$P^\pi(s'|s) = \sum_{a\in\mathcal{A}} \pi(a|s) P(s'|s,a) \tag{2}$$

$$R^\pi(s) = \mathbb{E}_\pi[R|s_t = s] = \sum_{a\in\mathcal{A}} \pi(a|s) r(s,a) \tag{3}$$

For a policy π let us define the Bellman operator $B^\pi : \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^{|\mathcal{S}|}$ by $B^\pi U = R^\pi + \gamma P^\pi U$ for any $U \in \mathbb{R}^{|\mathcal{S}|}$. It is easy to prove [10] that, for $\gamma < 1$, the Bellman operator is a strict contraction map and that its only fixed point is V^π.

Let us define the Bellman optimality backup operator $B^* : \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^{|\mathcal{S}|}$ by $(B^*U)(s) = \max_{a\in\mathcal{A}} \big(r(s,a) + \gamma P(\cdot|s,a)U\big)$ for any $U \in \mathbb{R}^{|\mathcal{S}|}$ and $s \in \mathcal{S}$. Similarly, for $\gamma < 1$, the Bellman optimality backup operator is a strict contraction map. By denoting V^* its only fixed point and Π the set of all policies, it comes that $\forall s \in \mathcal{S}, V^*(s) = \max_{\pi\in\Pi} V^\pi(s)$ and there exists an optimal policy π^* (even a stationary one) such that $V^{\pi^*}(s) = V^*(s)$

1.2.4 State-action value function and Bellman operator

Similarly to the value function, for any policy π, we can define the state-action value function Q by:

$$Q^\pi(s,a) = \mathbb{E}_\pi[G_t|s_t = s, a_t = a] \tag{4}$$

where $\mathbb{E}_\pi$ is the expected-value under the assumption that the agent follows policy π at each time step. Defining the optimal Q-function by

$$Q^*(s,a) = \max_{\pi\in\Pi} Q^\pi(s,a) \tag{5}$$

it comes that:

$$\forall s \in \mathcal{S}, V^*(s) = \max_{a\in\mathcal{A}} Q^*(s,a) \tag{6}$$

and the policy:

$$\pi^*(s) = \operatorname*{argmax}_{a\in\mathcal{A}} Q^*(s,a) \tag{7}$$

is optimal.

Finally, we have the Bellman Equation for the optimal Q-function:

$$Q^*(s,a) = r(s,a) + \gamma \sum_{s'\in\mathcal{S}} P(s'|s,a) \max_{a'\in\mathcal{A}} Q^*(s',a') \tag{8}$$

2 Single-agent learning algorithms

Many approaches exist to learn optimal or near-optimal policies in MDPs. All have their own pros and cons depending on the type of environment or the learning objective for the agent. In this section, we are going to introduce two algorithms for single-agent reinforcement learning that will be necessary to the study of multi-agent scenarios.

Regarding the elements introduced above, a perfect knowledge of the environment (i.e. P and r functions) allows us to compute dynamically an optimal policy π^*. However, our learning task is made in an unknown environment. Thus, our objective is to approach an optimal policy without this knowledge of the MDP dynamics. Such algorithms can be classified into two main groups: *model-based* and *model-free*. In *model-based* reinforcement learning, we try to maintain a knowledge of the model (transition and rewards dynamics) from which an optimal policy can be derived. On the other hand, *model-free* reinforcement learning uses functions other than P and r from which it is possible to compute nearly-optimal policies.

2.1 Model-free reinforcement learning: Q-Learning

This section introduces one of the most commonly used *model-free* reinforcement learning algorithms: Q-learning. As shown before (Equation 7), the state-action value function contains all the information to compute an optimal deterministic policy.

The offline Q-learning algorithm (Algorithm 1 with a fixed learning rate) relies on the Bellman Equation (8) to update successively the value of the Q-function until convergence. Starting from a list of observed state-action-reward $(s_0, a_0, r_0, ..., s_n, a_n, r_n)$, the Q-function is updated with the rule:

$$Q_{t+1}(s_t, a_t) = (1 - \alpha_t(s_t, a_t))Q_t(s_t, a_t) + \alpha_t(s_t, a_t)(r_t + \gamma \max_{a' \in \mathcal{A}} Q_t(s_{t+1}, a')) \tag{9}$$

The fundamental result of the Q-learning algorithm (proved for example in [7]) is the following:

Theorem 2.1. *Under the following assumptions:*

$$\forall (s, a) \in \mathcal{S} \times \mathcal{A}, \quad \sum_{t=0}^{+\infty} \alpha_t(s, a) = +\infty \tag{10}$$

$$\forall (s, a) \in \mathcal{S} \times \mathcal{A}, \quad \sum_{t=0}^{+\infty} \alpha_t(s, a)^2 < +\infty \tag{11}$$

the Q-learning algorithm converges almost surely to the optimal Q-function.

Algorithm 1 Offline Q-learning algorithm

Require: $\mathcal{S}$, $\mathcal{A}$, learning rate $\alpha \in [0, 1]$, discount factor $\gamma \in [0, 1]$, sequence $(s_0, a_0, r_0, ..., s_n, a_n, r_n)$
  Initialize $Q(s, a)$ arbitrarily for any state-action pair $(s, a) \in \mathcal{S} \times \mathcal{A}$
  for $t = 0, ..., n$ **do**
    $Q(s_t, a_t) \leftarrow (1 - \alpha) \cdot Q(s_t, a_t) + \alpha \cdot (r_t + \gamma \cdot \max_{a'} Q(s_{t+1}, a'))$
  end for
  return Q

As mentioned, the former algorithm is an *offline-learning* algorithm. Concretely, it means that the learning task is based on actions (and observations) that the agent was not able to control. To this extent, the agent requires an *exploration policy*. Choosing a random policy is a first possibility to guarantee that each state-action pair is visited infinitely often (assumption (10)). However, this algorithm would imply bad results in terms of cumulative reward by playing a sub-optimal exploration policy. One possible alternative is the commonly used ϵ-greedy exploration (Algorithm 2 with constant ϵ value). At each step t, with probability ϵ_t, the agent plays a random action. Otherwise, the agent plays the *optimal action relative to the current Q-function*: $a_t = \operatorname{argmax}_{a \in \mathcal{A}} Q_t(s_t, a)$.

The theoretical guarantee for convergence of the Q-function ensures that, *after enough time*, the computed optimal policy will be the true optimal policy. However, the algorithm delivers no theoretical guarantees on the cumulative reward obtained during the learning phase. If the objective is to combine an efficient learning and an acceptable reward during the learning, new types of algorithms will be necessary.

2.2 Near-optimal reinforcement learning

In this section, it is assumed that the agent learns during a given number m of episodes of length T. At the beginning of each episode i, the agent restarts in a state s_i (possibly randomly chosen or always the same). At this point, when doing *online learning* (the agent learns in an environment in which it can choose how to behave), it is often interesting for an agent not only to learn a good final policy but also to accumulate good rewards. Consequently, at each time step, the algorithms needs to strike a balance between playing what currently seems to be the optimal action according to its knowledge of the environment and trying a new or unknown action to explore the environment. This dilemma is know as the *exploration-exploitation tradeoff*.

Algorithm 2 Online Q-learning algorithm with ϵ-greedy exploration strategy

Require: $\mathcal{S}$, $\mathcal{A}$, learning rate $\alpha \in [0,1]$, discount factor $\gamma \in [0,1]$, sequence $(s_0, a_0, r_0, ..., s_n, a_n, r_n)$
  Initialize $Q(s,a)$ arbitrarily for any state-action pair $(s,a) \in \mathcal{S} \times \mathcal{A}$
  while Q is not converged **do**
    Start in state $s \in \mathcal{S}$
    while s is not terminal **do**
      Calculate π according to Q and exploration strategy (e.g. $\pi(x) \leftarrow$ argmax $_a Q(x,a)$)
      $a \leftarrow \pi(s)$
      Receive reward r and new state s'
      $Q(s,a) \leftarrow (1-\alpha) \cdot Q(s,a) + \alpha \cdot (r + \gamma \cdot \max_{a'} Q(s',a'))$
      $s \leftarrow s'$
    end while
  end while
  return Q

2.2.1 Regret

In order to measure the quality of an online learning algorithm, let us introduce the common *regret* that measures, for all episodes, the difference between the best expected reward and the real reward:

$$L_m = \sum_{i=1}^{m} \left(V^*(s_0^{(i)}) - \sum_{t=1}^{T} R_t^{(i)} \right) \tag{12}$$

If the agent plays an optimal policy, the following holds:

$$\mathbb{E}[L_m] = 0 \tag{13}$$

$$\frac{L_m}{m} \underset{m \to +\infty}{\longrightarrow} 0 \quad \text{with probability 1} \tag{14}$$

More generally, an agent is considered to achieve a good online learning if it gets a *sublinear regret* which can be achieved by the strong assumption:

$$L_m = o_{m \to +\infty}(m) \quad \text{with probability 1} \tag{15}$$

or the weaker one:

$$\mathbb{E}\left[\frac{L_m}{m}\right] \underset{m \to 0}{\longrightarrow} 0 \tag{16}$$

In this part, two major single-agent algorithms achieving sublinear regret will be introduced. Although other algorithms achieve similar results (sometimes better in special contexts), these two algorithms are very commonly used and will present great interest in the multi-agent approach.

2.2.2 Near-optimal reinforcement learning with upper confidence bounds: URCL2 algorithm

The first algorithm, UCRL2 [2], uses the very famous principle in reinforcement learning of *Optimism in the face of uncertainty* designed to deal with the *exploration-exploitation tradeoff*. The idea is that, at each time step, the agent has an approximate knowledge of the underlying MDP. A *greedy* approach would consist in choosing each time the optimal action regarding the computed MDP. Instead, the *Optimism in the face of uncertainty* approach consists in computing, for each state-action pair, an upper confidence bound on the Q-function.

This approach can be easily illustrated in the case of a *stochastic bandit*, which is a MDP with a single state ($|\mathcal{S}| = 1$). Let us assume that, for every action, the reward is within $[0,1]$. Then, for any action $a \in \mathcal{A}$, if $r(a)$ is the true expected reward and $\bar{r}_a$ is the empirical mean

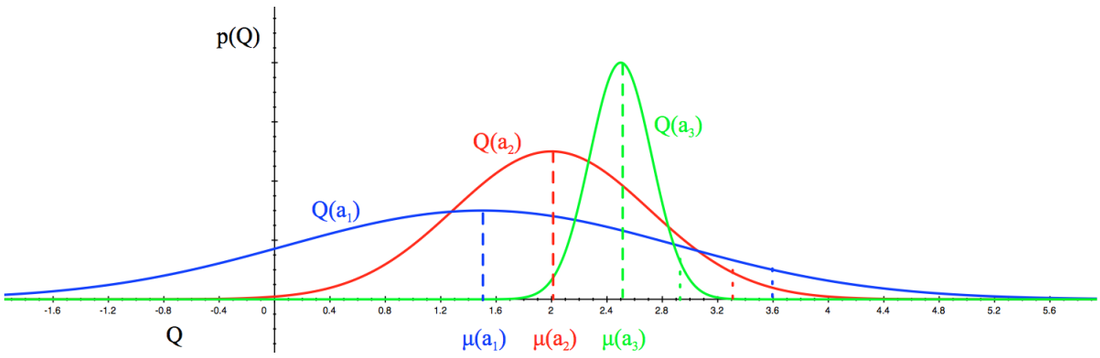

Figure 2: Upper-confidence bound for a stochastic bandit

after playing action a n times, then, Hoeffding's inequality ensures that, with probability at least $1-\delta$:

$$|\bar{r}_a - r(a)| \leq \sqrt{\frac{1}{2n}\log\left(\frac{2}{\delta}\right)} \tag{17}$$

Consequently, with δ-confidence, the true mean reward $r(a)$ falls within the confidence interval $\left[\bar{r}_a - \sqrt{\frac{1}{2n}\log\left(\frac{2}{\delta}\right)}, \bar{r}_a + \sqrt{\frac{1}{2n}\log\left(\frac{2}{\delta}\right)}\right]$, which allows us to consider, for each action a, the *upper confidence bound* (or optimistic reward) $\bar{Q}(a) = \bar{r}_a + \sqrt{\frac{1}{2n}\log\left(\frac{2}{\delta}\right)}$. At each time step, the agent chooses the action with the highest upper confidence bound. In the example (Figure 2), the true optimal action is a_3. The current maximum upper confidence bound corresponds to a_1 that has been played less than other actions. After a few time steps, the mean distribution will become peaky until $\bar{Q}(a_1) < \bar{Q}(a_2)$. Action a_2 will then have the highest upper confidence bound and will be played until the UCB is decreased enough so that $\bar{Q}(a_2) < \bar{Q}(a_3)$. The algorithm will then converge to a_3 as optimal action.

Let us assume that the algorithm is run for m steps. Let us denote $S = |\mathcal{S}|$ and $A = |\mathcal{A}|$. Let us also introduce D, the diameter of the MDP defined as:

$$D = \max_{s,s' \in \mathcal{S}, s \neq s'} \min_{\pi \in \Pi} \mathbb{E}\left[T(s'|\pi, s)\right] \tag{18}$$

where $T(s'|\pi, s)$ is the random variable for the first time step in which state s' is reached in this process starting from state s and applying policy π.

Then, the following theorem gives a theoretical guarantee of sublinear regret (in terms of expected value, cf. strong hypothesis (15)):

Theorem 2.2. *With probability at least* $1-\delta$*:*

$$L_m \leq 34DS\sqrt{Am\log\frac{m}{\delta}}$$

2.2.3 Near-optimal reinforcement learning with Bayesian statistics: Thompson Sampling

Like UCRL2, many algorithms (E^3, R-max) deal with the *exploration-exploitation tradeoff* introducing optimism about poorly-understood state-action pairs. Thompson Sampling is an alternative approach to the paradigma of *Optimism in the face of uncertainty*. The PSRL algorithm [8] introduces an alternative approach for an efficient exploration via *posterior sampling*. This method uses Bayesian statistics through a *posterior probability* for the true parameters of the underlying Markov Decision Process. At each time step, the agent samples a MDP from the posterior distribution and the history of observations. It is then able to solve this MDP by dynamic programming and compute its optimal policy.

Let us assume that the algorithm is run for m episodes of length τ. Let M^* be the true underlying MDP, $S = |\mathcal{S}|$ and $A = |\mathcal{A}|$. Then, the following theorem gives a theoretical guarantee of sublinear regret (in terms of expected value, cf. weaker hypothesis (16)):

Theorem 2.3. *If* f *is the distribution of* M^**:*

$$\mathbb{E}[L_m] = O\left(\tau S\sqrt{Am\log(SAm)}\right)$$

Algorithm 3 UCRL2 algorithm

Require: $\mathcal{S}$, $\mathcal{A}$, confidence parameter δ

Initialization: Set $t := 1$, and observe the initial state s_1.

for episodes $k = 1, 2, ...$ **do**

Initialize episode k

For all $(s, a) \in \mathcal{S} \times \mathcal{A}$ initialize the state-action counts for episode k, $v_k(s, a) = 0$.

Set the state-action counts prior to episode k, $N_k(s, a) = |\{\tau < t_k : s_\tau = s, a_\tau = a\}|$ where t_k is the start time of episode k.

For all $(s, a) \in \mathcal{S} \times \mathcal{A}$, set the observed accumulated rewards and the transition counts prior to episode k:

$$R_k(s, a) = \sum_{\tau=1}^{t_k-1} r_\tau \mathbb{1}_{s_\tau=s, a_\tau=a}$$

$$P_k(s, a, s') = |\{\tau < t_k : s_\tau = s, a_\tau = a, s_{\tau+1} = s'\}|$$

Compute estimated mean reward and transition probabilities:

$$\hat{r}_k(s, a) = \frac{R_k(s, a)}{\max(1, N_k(s, a))}$$

$$\hat{p}_k(s, a, s') = \frac{P_k(s, a, s')}{\max(1, N_k(s, a))}$$

Compute policy $\tilde{\pi}_k$

Let $\mathcal{M}_k$ be the set of MDPs of states and actions sets $\mathcal{S}$ and $\mathcal{A}$ with mean rewards $\tilde{r}$ and transition probabilities $\tilde{p}$ close enough to $\hat{r}_k$ and $\hat{p}_k$ so that:

$$|\tilde{r}(s, a) - \hat{r}_k(s, a)| \leq \sqrt{\frac{7 \log\left(\frac{2|\mathcal{S}||\mathcal{A}|t_k}{\delta}\right)}{2 \max(1, N_k(s, a))}}$$

$$||\tilde{p}(\cdot|s, a) - \hat{p}_k(s, a, \cdot)||_1 \leq \sqrt{\frac{14|\mathcal{S}| \log\left(\frac{2|\mathcal{A}|t_k}{\delta}\right)}{2 \max(1, N_k(s, a))}}$$

Compute the most optimistic MDP $\tilde{M}_k$ within $\mathcal{M}_k$ and policy $\tilde{\pi}_k$ (cf. Value iteration algorithm *in* [2]).

Compute policy $\tilde{\pi}_k$

while $v_k(s_t, \tilde{\pi}_k(s_t)) < \max(1, N_k(s_t, \tilde{\pi}_k(s_t)))$ **do**

Choose action $a_t = \tilde{\pi}_k(s_t)$, obtain reward r_t , and observe next state s_{t+1}.

Increment $v_k(s_t, \tilde{\pi}_k(s_t))$

end while

end for

Algorithm 4 PSRL algorithm

Require: Prior distribution f, horizon τ

for episodes $k = 1, 2, ...$ **do**

Sample the MDP $M_k \sim f(\cdot|H_k)$ where H_k is the history until episode k.

Solve the MDP via dynamic programming and compute the optimal policy $\pi_k = \pi^{*M_k}$

for timesteps $t = 1, 2, ...$ **do**

Choose action $a_t^{(k)} \sim \pi_k^{(t)}(s_t^{(k)})$

Apply action $a_t^{(k)}$, observe state $s_{t+1}^{(k)}$ and collect reward $r_t^{(k)}$

end for

end for

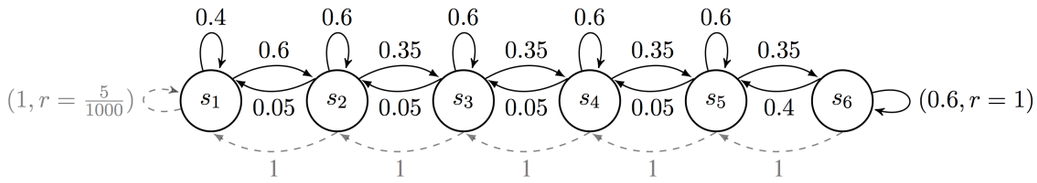

Figure 3: *RiverSwim* MDP with normalized rewards

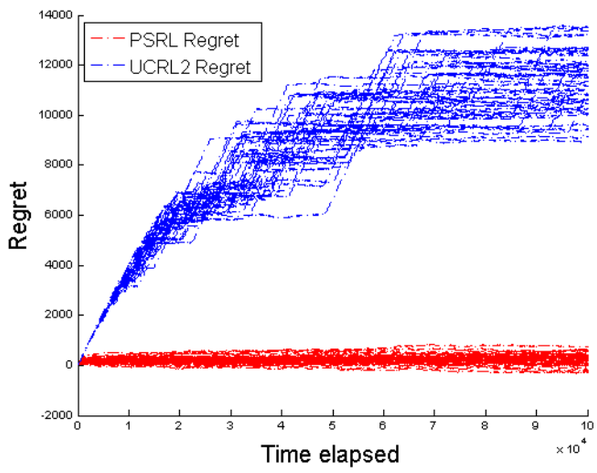

Figure 4: Simulated regret on the ∞-horizon RiverSwim environment from [8]

2.2.4 Near-optimal reinforcement learnings: Comparing UCRL2 and PSRL

As mentioned in the previous sections, UCRL2 and PSRL are both able to achieve sublinear regret. However, on could wonder how they compete with each other in real scenarios. To this extent, let us consider the standard environment called *RiverSwim* introduced in [11] (Figure 3).

The *RiverSwim* MDP is discribed in [11]:

> RiverSwim consists of six states. The agent starts in one of the states near the beginning of the row. The two actions available to the agent are to swim left or right. Swimming to the right (against the current of the river) will more often than not leave the agent in the same state, but will sometimes transition the agent to the right (and with a much smaller probability to the left). Swimming to the left (with the current) always succeeds in moving the agent to the left, until the leftmost state is reached at which point swimming to the left yields a small reward of five units. The agent receives a much larger reward, of ten thousand units, for swimming upstream and reaching the rightmost state. This MDP requires sequences of actions in order to explore effectively

UCRL2 (and other *optimisme in the face of uncertainty* - OFU algorithms) and PSRL have a similar starting approach. Either by confidence sets (OFU) or by statistical plausibility (PS), all these algorithms tend to describe, either with high probability or through probability distributions, the set of possible underlying MDPs. Whereas UCRL2 then chooses an optimal policy by maximizing value simultaneously over policies and MDPs in this set (which is computationally demanding, even with the *value-iteration* algorithm), PSRL only solves one sampled MDP. Figure 4, compares the results of the two algorithms where the priors for transitions and rewards are expressed in terms of Dirichlet and normal-gamma distributions. The results suggest that Thompson Sampling achieves better results by obtaining faster convergence. Without going deep into the theoretical guarantees of the better performance of this approach, let us mention that, in [9], Osband and Van Roy give arguments justifying the interest of priors over upper bound to guarantee a higher proximity of the optimal policies and, thus, a lower regret.

As a conclusion, let us insist on two main points. PSRL gives a less intuitive approach that OFU and is very dependent on the chosen prior. However, when the prior is well-

adapted, it leads to a more efficient learning and less computing time.

3 Multi-agent reinforcement learning overview

3.1 Multi-agent reinforcement learning framework

In this section, let us introduce briefly the general framwork for the multi-agent case derived from the single-agent framework. Our framework makes two initial assumptions:

- *Global observation hypothesis*: at each moment, every agent i can observe the global state $\mathbf{s}$
- *No communication hypothesis*: at each transition, an agent i can only observe his own action a_i

Let us motivate the second hypothesis (*no communication hypothesis*). It can be interpreted in two ways. Firstly, this framework can be extended to the *true no communication* or *true no action observation* case in which it is truly impossible for an agent to observe the actions of other players. Secondly, this framework can also be used in scenarios in which actions are observable, but in which, for computational reasons with too many players, it is not reasonable to do tabular learning on the joint action space which grows exponentially with the number of players. Thirdly, let us bring froward two arguments to motivate the difference between the observability of the joint-state set and the joint-action set. Firstly, although it is the core of multi-agent reinforcement learning to have agents playing their own actions, many scenarios truly rely on a *real* common state. Also, it is notable that *state-approximation* through neural networks can be adopted to fit our framework without growing exponentially with the number of players.

Finally, let us introduce our additional hypothesis of *cooperative framework* : all agents share the same reward. At this point, it can be mentioned that the cooperation hypothesis can be more general with agents obtaining different rewards and trying to maximize their aggregated reward. This is not the hypothesis used here.

Let us model the MARL problem with n agents as a Markov Decision Problem $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, r \rangle$. $\mathcal{S}$ is the finite state space. $\mathcal{A} = \mathcal{A}_1 \times ... \times \mathcal{A}_n$ is the finite joint action space. The stationary markovian transition dynamics assumption gives: $P(\mathbf{s}_{t+1}|\mathbf{s}_0, \mathbf{a}_0, \ldots, \mathbf{s}_t, \mathbf{a}_t) = P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$. Finally, let us denote $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ the reward function such that $r(\mathbf{s}, \mathbf{a}) = \mathbb{E}(R(\mathbf{s}, \mathbf{a}))$.

3.2 Distributed Q-learning algorithms

As seen in (2.1), the Q-function is an essential element in single-agent reinforcement learning that gives rise to a wide class of algorithms. The Q-function still remains well defined and relevant in multi-agent reinforcement learning (section 3.2.1). However, since the Q-function is focused on a global approach of the environment (joint spaces $\mathcal{S}$ and $\mathcal{A}$), Q-learning algorithm is not directly applicable to the multi-agent framework and will require the introduction of the max-Q function (section 3.2.2).

3.2.1 Policies and Q-function in multi-agent reinforcement learning

In this section, let us introduce $\mathbf{\Pi}$ the set of all *joint policies* ($\mathbf{\Pi} = \{\boldsymbol{\pi} : \mathcal{S} \to \mathcal{A}\}$). For any agent i, let Π_i denote the set of agent i's policies ($\Pi_i = \{\pi_i : \mathcal{S} \to \mathcal{A}_i\}$). In particular, $\mathbf{\Pi} = \Pi_1 \times ... \times \Pi_n$.

In the multi agent-case, the Q-function and the optimal Q-function are extend by the following definition: for any joint policy $\boldsymbol{\pi} \in \mathbf{\Pi}$, for any $(\mathbf{s}, \mathbf{a}) \in \mathcal{S} \times \mathcal{A}$:

$$Q^{\boldsymbol{\pi}}(\mathbf{s}, \mathbf{a}) = \mathbb{E}_{\boldsymbol{\pi}}[G_t | \mathbf{s}_t = \mathbf{s}, \mathbf{a}_t = \mathbf{s}] \tag{19}$$

where $\mathbb{E}_{\boldsymbol{\pi}}$ is the expected-value under the assumption that the agents follow policy $\boldsymbol{\pi}$ at each time step. Similarly, the optimal Q-function is defined by:

$$Q^*(\mathbf{s}, \mathbf{a}) = \max_{\boldsymbol{\pi} \in \mathbf{\Pi}} Q^{\boldsymbol{\pi}}(\mathbf{s}, \mathbf{a}) \tag{20}$$

3.2.2 Max-Q function

In multi-agent reinforcement learning, the objective is not to compute directly an optimal joint policy $\boldsymbol{\pi} \in \boldsymbol{\Pi}$ but rather, for every agent i, to compute a policy $\pi_i \in \Pi_i$ such that the joint policy $(\pi_1, ..., \pi_n)$ is optimal. For this reason, since other players' actions are not observable, the Q-function can neither be computed or used. Instead, how to define a relevant and useful variant of the Q-function that would not depend on a joint policy $\boldsymbol{\pi}$ and joint action $\mathbf{a}$ but rather of marginal policy π_i and marginal action a_i?

To this extent, [3] introduces the max-Q function. Firstly, for an agent i, a marginal policy π, a joint state $\mathbf{s}$ and a marginal action a_i, let us define:

$$q_i^{\pi}(\mathbf{s}, a) = \max_{\boldsymbol{\pi} \in \boldsymbol{\Pi} \text{ s.t. } \boldsymbol{\pi}_i = \pi} \max_{\mathbf{a} \in \mathcal{A} \text{ s.t. } \mathbf{a}_i = a} Q^{\boldsymbol{\pi}}(\mathbf{s}, \mathbf{a}) \tag{21}$$

The value $q_i^{\pi}(\mathbf{s}, a)$ represents the best possible reward when agent i plays policy π, starts with joint state-action $(\mathbf{s}, a)$ and other players play the best possible initial action and policies corresponding to the triplet $(\pi, \mathbf{s}, a)$.

Finally, for a joint state-action $(\mathbf{s}, a)$, let us define the max-Q function:

$$q_i(\mathbf{s}, a) = \max_{\pi \in \Pi_i} q_i^{\pi}(\mathbf{s}, a) \tag{22}$$

Similarly, the value $q_i^{\pi}(\mathbf{s}, a)$ represents the best possible reward when agent i starts with joint state-action $(\mathbf{s}, a)$ and all players (including agent i) play the best possible initial action and policies corresponding to the joint state-action pair $(\mathbf{s}, a)$.

The interest of this max-Q function essentially comes from this property:

Proposition 3.1. *For any given agent i and state $\mathbf{s} \in \mathcal{S}$, if $\hat{\mathbf{a}} \in \mathcal{A}$ is such that $Q^*(\mathbf{s}, \hat{\mathbf{a}}) = \max_{\mathbf{a} \in \mathcal{A}} Q^*(\mathbf{s}, \mathbf{a})$, then:*

$$q_i(\mathbf{s}, \hat{\mathbf{a}})_i = \max_{a \in \mathcal{A}_i} q_i(\mathbf{s}, a) \tag{23}$$

Proposition 3.1 shows the interest of the max-Q function. If a joint action is optimal, then all marginal actions are optimal with respect to the max-Q function. Is the converse proposition true? Not in general. However, it is easy to see (see for example [3]) that the converse proposition is true as soon as there is only a single optimal joint action (Proposition 3.2). On the other hand, if there are several optimal joint actions for a given state, players can face the *problem of coordination* (see [3]).

Proposition 3.2. *Let us make the assumption that, for a given state $\mathbf{s} \in \mathcal{S}$, there is a unique $\hat{\mathbf{a}} \in \mathcal{A}$ such that $Q^*(\mathbf{s}, \hat{\mathbf{a}}) = \max_{\mathbf{a} \in \mathcal{A}} Q^*(\mathbf{s}, \mathbf{a})$. Then, for every agent i, there is a unique $a_i \in \mathcal{A}_i$ such that $q_i(\mathbf{s}, a_i) = \max_{a \in \mathcal{A}_i} q_i(\mathbf{s}, a)$ and:*

$$\forall i, q_i(\mathbf{s}, a_i) = Q^*(\mathbf{s}, \hat{\mathbf{a}}) \tag{24}$$

$$\hat{\mathbf{a}} = (a_1, ..., a_n) \tag{25}$$

3.2.3 Distributed Reinforcement learning (max-Q algorithm)

Section 3.2.2, in particular propositions (3.1) and (3.2) show the interest of the max-Q function. In an environment in which every state has a single optimal joint action (see assumption of Proposition 3.2), if every agent is able to compute its max-Q table, then, they are able to jointly play the optimal joint action. Is is thus natural to look for an algorithm aiming at computing the true max-Q function.

In [3], Lauer and Riedmiller present a Max-Q algorithm (Algorithm 5) that is able to achieve coordination in deterministic environments.

Theorem 3.3. *Let us assume that the environment is deterministic (for every state action pair, the reward and the transition are deterministic) and that the rewards are always non negative. If every state-action pair occurs infinitely often, then the q-function monotonically converges to the true max-Q function and the joint policy $(\pi_i)_{0 \leq i \leq n}$ converge to a joint policy $\boldsymbol{\pi}^*$ that is optimal.*

Algorithm 5 Offline Q-learning algorithm for agent i from [3]

Require: $\mathcal{S}$, $\mathcal{A}_i$, discount factor $\gamma \in [0,1]$, sequence $(\mathbf{s}_0, a_0, r_0, ..., \mathbf{s}_n, a_n, r_n)$
  Initialize $q_i(\mathbf{s}, a)$ arbitrarily for any state-action pair $(\mathbf{s}, a) \in \mathcal{S} \times \mathcal{A}_i$
  Initialize $\pi_i(\mathbf{s}) \in \mathcal{A}_i$ arbitrarily for any state $\mathbf{s} \in \mathcal{S}$
  for $t = 0, ..., n$ **do**
    $q(\mathbf{s}_t, a_t) \leftarrow$ max $(q(\mathbf{s}_t, a_t), r_t + \gamma \cdot \max_{a'} q(\mathbf{s}_{t+1}, a'))$
    If $q(\mathbf{s}_t, a_t)$ was modified $(r_t + \gamma \cdot \max_{a'} q(\mathbf{s}_{t+1}, a' > q(\mathbf{s}_t, a_t))$, update $\pi_i(\mathbf{s}) \leftarrow a_t$
  end for
  return q_i and π_i

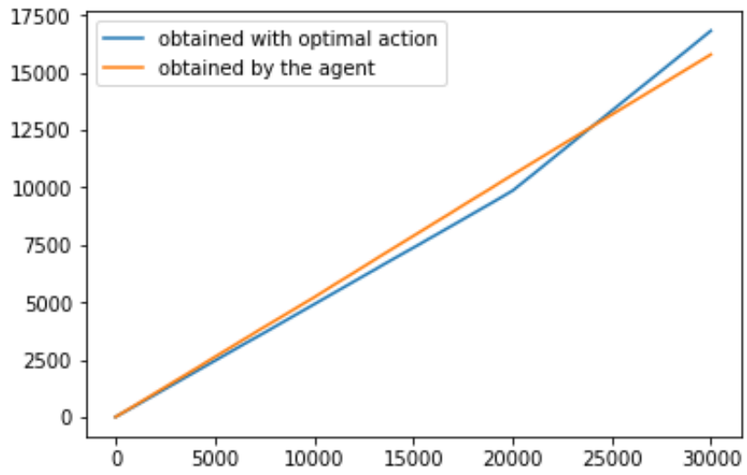

Figure 5: Max-Q algorithm in a Bandit with 2 players and 10 actions

If this results gives a good guarantee in a deterministic framework, what can we expect from a stochastic scenario? To understand how it behaves, the max-Q algorithm was tested in a simple bandit with two players and 10 actions each (Figure 5). During the first 20000 steps (*learning phase)*, the other player adopts a random policy. The agent is forced to learn by playing randomly). During the last 10000 steps, the other agent plays the optimal (cooperative) action. The cumulative reward is compared to another agent playing the optimal action all the time.

The results (Figure 5), especially the final slope of the curve, show that the max-Q algorithm fails to complete offline learning in spite of the infinite state-action pair visit assumption. To understand these results, it is interesting to look at the curve of the max-Q functions for the true optimal action as well as in the case of the maximum computed max-Q. These observations allow us to notice that the max-Q function tends to *overshoot* the true max-Q value. This is due to the fact that the algorithm is designed for deterministic frameworks and will consider every high reward as a *mean value* for a state-action pair instead as a single value geenrated by a distribution.

Theorem 3.4. *Under the assumption that every state-action pair* (s, a) *observed infinitely often, no algorithm can compute the max-Q function.*

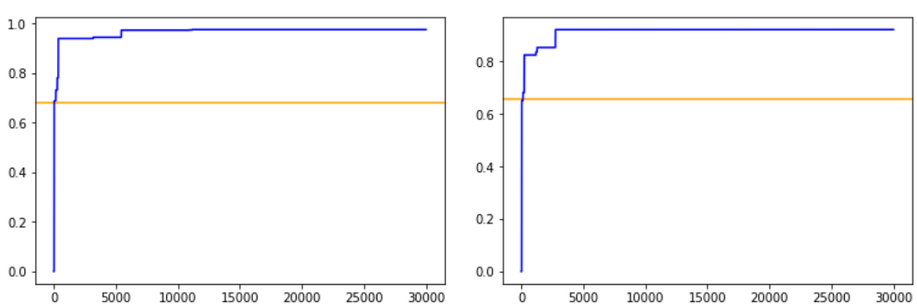

Figure 6: Computed max-Q functions for the true optimal action (left) and the computed max-Q (right). For these two actions, the blue curve represents the computed max-Q function and the yellow line represents the true max-Q

	a_1	a_2
a_0 (`BANDIT1`)	$\delta(0)$	$\delta(1)$
a_0 (`BANDIT2`)	$Bernouilli(1/2)$	$Bernouilli(1/2)$

Figure 7: Example Bandits

Proof. Let us consider the following bandits (`BANDIT1` and `BANDIT2` Figure 7) with 1 action a_0 for player 1 and 2 actions a_1, a_2 for player 2. The table gives the distribution of the rewards for the joint action pairs (a_0, a_1) and (a_0, a_2).

If player 2 plays a perfect random policy, the distributions of rewards will be the same for the 2 bandits: $Bernouilli(1/2)$ whereas $q_{Bandit1}(a_0) = 1 \neq \frac{1}{2} = q_{Bandit2}(a_0)$

□

3.2.4 Hysteretic-Q

To deal with the problem of *overshooting* in stochastic frameworks as well as to offer an implicit solution to the problem of coordination (off-policy learning), Matignon et al. [5] introduced the Hysteretic-Q algorithm (Algorithm 6). The max-Q algorithms works by brusquely increasing the value of the max-Q function each time the *pseudo-gradient* $r_t + \gamma \max_{a \in \mathcal{A}_i} q_i(\mathbf{s}_{t+1}, a) - q_i(\mathbf{s}_t, a_i)$ becomes positive. This property allows the algorithm to converge very fast in deterministic frameworks. However, this is also the reason why the algorithm cannot achieve agents coordination with off-policy learning and also fails in a stochastic context (*overshoots*). In order to compensate, the Hysteretic-Q learning modifies the value of the max-Q function each time, by increasing it as well as decreasing it. The key idea is to use two different learning rates: the *increase learning rate* α and the *decrease learning rate* β, with $\alpha \gg \beta$ to encourage increases and compute the max-Q.

Algorithm 6 Offline Hysteretic-Q algorithm for agent i from [5]

Require: $\mathcal{S}$, $\mathcal{A}_i$, learning rates α and β, discount factor $\gamma \in [0, 1]$, sequence $(\mathbf{s}_0, a_0, r_0, ..., \mathbf{s}_n, a_n, r_n)$
  Initialize $q_i(\mathbf{s}, a)$ arbitrarily for any state-action pair $(\mathbf{s}, a) \in \mathcal{S} \times \mathcal{A}_i$
  for $t = 0, ..., n$ **do**
    Define $\delta := r_t + \gamma \max_{a \in \mathcal{A}_i} q_i(\mathbf{s}_{t+1}, a) - q_i(\mathbf{s}_t, a_i)$
    if $\delta > 0$ **then**
      $q(\mathbf{s}_t, a_t) \leftarrow q(\mathbf{s}_t, a_t) + \alpha\delta$
    else
      $q(\mathbf{s}_t, a_t) \leftarrow q(\mathbf{s}_t, a_t) + \beta\delta$
    end if
  end for
  return q_i

If this algorithms is often very effective, it does not have any theoretical guarantees and it highly depends on the choice of learning rates α and β. To understand how it behaves, it was tested on the same bandit as before (Figure 9). During the first 20000 steps (*learning phase)*, the other player adopts a random policy. The agent is forced to learn by playing randomly). During the other 40000 steps, the other agent plays the optimal (cooperative) action. Contrary to the max-Q algorithm, the Hysteretic-Q algorithm is still free to change its strategy since its max-Q functions can also decrease. The cumulative reward is compared to another agent playing the optimal action all the time.

The results (Figure 9), especially the final slope of the curve, show that the hysteretic-Q algorithm fails to complete offline learning but performs significantly better than the max-Q thanks to the fact that it keeps learning all the time and does not overshoot the true max-Q as much as the max-Q algorithm. However, the algorithm is not fully satisfying to learn the max-Q functions and can overshoots like the max-Q algorithm (computed max-Q) or undershoot (true max-Q).

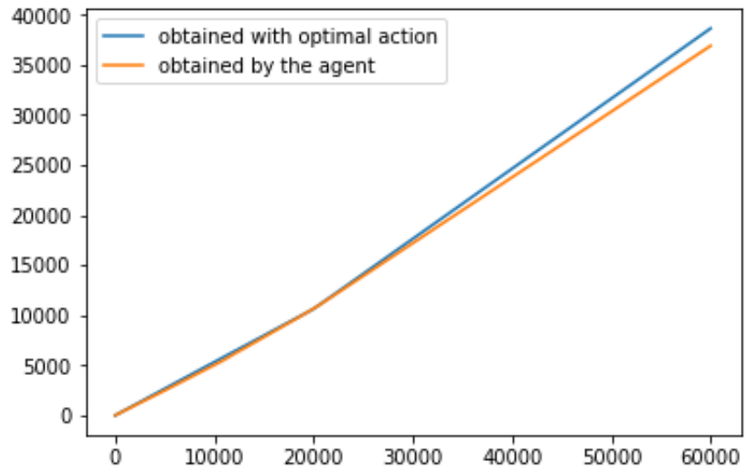

Figure 8: Hysteretic-Q algorithm in a Bandit with 2 players and 10 actions, $\alpha = 0.1$, $\beta = 0.01$

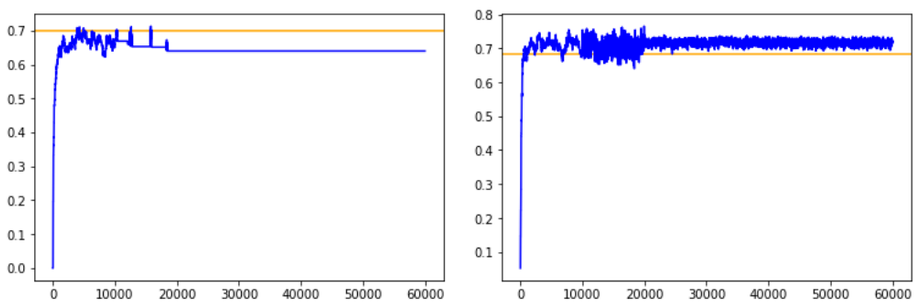

Figure 9: Computed max-Q functions for the true optimal action (left) and the computed max-Q (right). For these two actions, the blue curve represents the computed max-Q function and the yellow line represents the true max-Q

3.3 Near-optimal reinforcement learning in multi-agent scenarios

3.3.1 Regret in MARL: several approaches

Similarly to the single-agent case, it is important to introduce the multi-agent version of the regret in order to assess the quality of an *online learning* algorithm. However, contrarily to the single-agent case, the definition of the regret is not unique and various conceptions are equally relevant. Let us explain why. By definition, the regret is supposed to measure the difference between the cumulative reward obtained by an agent and the maximum reward that could have been expected. However, the definition of *best possible expected reward* is not as clear in a multi-agent scenario. As a matter of fact, if one tries to judge the behavior of one agent in an environment with other agents, it is necessary to take into account the fact that the agent might be limited in terms of maximum achievable reward by other agents' behaviors. Consequently, several definitions of the regrets are to be considered, depending on what one aims at assessing.

If the objective is to measure the quality of a distributed learning algorithm (all agents learning together in the environment), is seems natural to introduce as a regret the measure of the difference between the cumuative reward and the reward that wold have been obtained in average by *all agents playing the optimal joint policy* $\boldsymbol{\pi}^*$:

$$L_m = \sum_{i=i}^{m} \left(V^{\boldsymbol{\pi}^*}(s_0^{(i)}) - \sum_{t=1}^{T} R_t^{(i)} \right) \tag{26}$$

On the other hand, if one focuses on the learning one single agent in a multi-agent context, it is necessary to take into account how other agents can limit the maximum reward. Let us make the assumption that other players are successively playing policies $\bar{\pi}_1, ..., \bar{\pi}_m$ during m episodes.

$$L_m = \max_{\pi \in \Pi} \sum_{i=i}^{m} \left(V^{(\pi, \bar{\pi}_i)}(s_0^{(i)}) - \sum_{t=1}^{T} R_t^{(i)} \right) \tag{27}$$

Finally, since we are focusing on cooperative learning, it is also natural to introduce a variant of the former definition.

$$L_m = \sum_{i=i}^{m} \left(V^{(\pi_{coop}, \bar{\pi}_i)}(s_0^{(i)}) - \sum_{t=1}^{T} R_t^{(i)} \right) \tag{28}$$

where π_{coop} is the cooperative policy of the agent (*i.e.* the marginal policy associated to the optimal joint policy $\boldsymbol{\pi}^*$).

3.3.2 Near-optimal reinforcement learning in adversarial bandits: EXP3 algorithm

Before moving forward into near-optimal learning for MDP, let us mention the EXP3 algorithm [1] that is able to solve our problem in the case of a Bandit.

Algorithm 7 EXP3 algorithm from [1]

Require: $\mathcal{A}_i$, total number of steps m
  Initialize $\eta = \sqrt{\frac{\log(A)}{(e-1)m\log(A)}}$ and $\beta = \eta A$.
  Initialize $p_1(a) = \frac{1}{A}$ for any $a \in \mathcal{A}$
  for $t = 0, ..., n$ **do**
    Define for any $s \leq t$ and $a \in \mathcal{A}$, $\tilde{r}_s(a) = \frac{r_s}{p_s(a)} \mathbb{1}_{a_s = a}$.
    Define for any $a \in \mathcal{A}$, $w_t(a) = \exp\left(\eta \sum_{s=1}^{t} \tilde{r}_s(a)\right)$.
    Define for any $a \in \mathcal{A}$, $p_t(a) = (1-\beta)\frac{w_t(a)}{\sum_{a'=1}^{A} w_t(a')} + \frac{\beta}{A}$.
    Sample $a_t \sim p_t$, execute a_t and collect r_t.
  end for
  return q_i

Theorem 3.5. *For any number of episodes m, let us define the regret as in (27):*

$$L_m = \max_{a \in \mathcal{A}} \sum_{i=i}^{m} (r^{(i)}(a) - R_m)$$

Then, EXP3 algorithm guarantees that:

$$L_m = \tilde{O}(\sqrt{mA\log(A)}) \tag{29}$$

3.3.3 Starting point: near-optimal reinforcement learning in adversarial scenarios

This section briefly presents the algorithm OLRM by Lim et al. [4] which is the starting point of the project. OLRM is an algorithm for adversarial reinforcement learning that achieves sublinear regret. The principle of the algorithm is to use *optimism in the face of uncertainty* with upper confidence bounds and to regularly test these bounds with so-called *consistency checks*. The initial idea of the project was to adapt this algorithm to a cooperative framework. Let us mention the difficulties initially encountered to transpose the algorithm from the adversarial to the cooperative case:

- Firstly, in [4], Lim et al. introduce a *weaker* definition of the regret. The definitions from (26) or (27) are particularly relevant for this adversarial case. In the adversarial case, (28) could have been adapted from the cooperative case by replacing a cooperative policy π_{coop} by an adversarial policy π_{adv} where π_{adv} is characterized by the equation:

$$\bar{\pi}_{adv} = \operatorname*{argmin}_{\bar{\pi} \in \bar{\Pi}} \mathbb{E}[V^{(\bar{\pi})*}(s)] \tag{30}$$

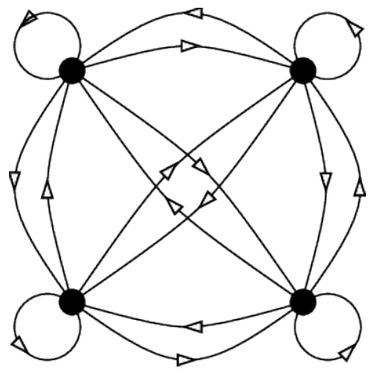

Figure 10: Well-mixed MAMDP

and π_{adv} is the optimal answer policy to $\bar{\pi}_{adv}$. However, the regret considered in [4] is defined as the difference between the optimal reward, assuming that the other agent plays an adversarial policy, and the obtained reward:

$$L'_m = \sum_{i=i}^{m} \left(V^{(\pi_{adv}, \bar{\pi}_{adv})}(s_0^{(i)}) - \sum_{t=1}^{T} R_t^{(i)} \right) \leq \sum_{i=i}^{m} \left(V^{(\pi_{adv}, \bar{\pi}_i)}(s_0^{(i)}) - \sum_{t=1}^{T} R_t^{(i)} \right) = L_m \tag{31}$$

It is thus a weaker definition for the regret that cannot be directly transposed to the cooperative case. Thus, trying to design an online learning algorithm for cooperative reinforcement learning is more challenging because it needs to compete with a more challenging regret.

- The other challenge in transposing adversarial learning to cooperative learning is the problem of exploration. The principle of OLRM is to use optimism in order to explore and decrease the upper confidence bound. On the other hand, the natural way of doing online learning in a cooperative context would be to use a pessimistic value for the Q-value and to increase the lower confidence bound. However, since more exploration leads to *increasing* the LCB instead of *decreasing* the UCB, this algorithm does not present a natural way to deal with the *exploration-exploitation tradeoff*. Since doing *offline learning* is not possible (Theorem 3.4), it is necessary to find a way of guiding the exploration for this new algorithm.

4 Cooperative case

Regarding the initial objective and the elements introduced before, two approaches seemed natural in our attempt to extend previous RL research to our cooperative framework. The first one consists in designing an algorithm to do distributed reinforcement learning (all agents learn together applying the same algorithm). The second one, closer to the initial approach in [4], consists in computing an algorithm to reach a sublinear regret in a cooperative case (given any succesive policies from other players, reach sublinear regret compared to the cooperative agent, see (28)).

4.1 Distributed algorithm for cooperative reinforcement learning in a stochastic environment

4.1.1 Well-mixed MAMDP

In this section, let us introduce the notion of *well-mixed MAMDP*. the idea of a *well-mixed chain* is to have states that communicate and rewards distributions that are smooth. Here, we introduce chains where transitions and rewards distributed this way:

For any state-action pair $(\mathbf{s}, \mathbf{a}) \in \mathcal{S} \times \mathcal{A}$:

- let us choose random mean and variance $\lambda_{s,a}, \sigma^2_{s,a} \in [0, 1]$. The reward distribution for this state action pair is a normal distribution: $R(\mathbf{s}, \mathbf{a}) \sim \mathcal{N}(\lambda_{\mathbf{s},\mathbf{a}}, \sigma^2_{\mathbf{s},\mathbf{a}})$
- the transitions probabilities are chosen by a Dirichlet distribution: $(p_{\mathbf{s},\mathbf{a},\mathbf{s}'})_{\mathbf{s}' \in \mathbf{S}} \sim Dir((1, ..., 1))$

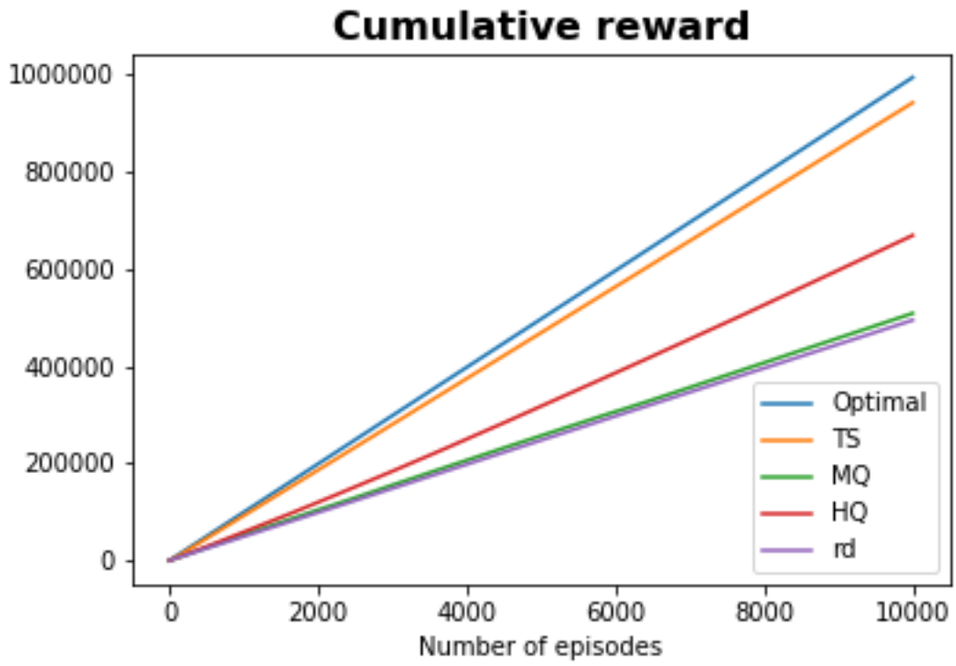

Figure 11: Max-Q, Hysteretic-Q and PSRL with Gamma-normal and Dirichlet distributions in a well-mixed MAMDP.

4.1.2 Thompson Sampling in a well-mixed MAMDP

Our first approach consisted in trying previously evoked and well-known algorithms for multi-agent reinforcement learning: max-Q and Hysteretic-Q. As expected, the results turned out to be disappointing given the relative complexity of a *stochastic* environment. In addition to these algorithms, Figure 11 shows the result of Thompson Sampling algorithm. Although it is initially designed as a SARL algorithm, the 10 players achieve a very good result. In a basic stochastic environment, players are still able to reach a near-optimum of coordination with PSRL algorithm.

4.2 Near-optimal regret learning: pessimistic-Q algorithm

4.2.1 Pessimistic-Q algorithm

In this section, let us suppose that, for state-action pair $(\mathbf{s}, a)$, depending on other players' actions, the reward mean and transition distribution belong to an uncertainty set $(r_{s,a}, p_{s,a}) \in \mathcal{U}^*(s,a)$. Let us assume that we know that this *true* uncertainty set $\mathcal{U}^*(s,a)$ is within a known class of *possible* uncertainty sets $\mathfrak{U}(s,a)$. This set is considered to be nested: for any $\mathcal{U}_1, \mathcal{U}_2 \in$, either $\mathcal{U}_1 \subset \mathcal{U}_2$ or $\mathcal{U}_2 \subset \mathcal{U}_1$.

Initially based on the max-Q algorithm and on OFU algorithms, the idea of the Pessimistic-Q algorithm is to maintain an equivalent of the Q-function (here, a belief on the uncertainty set $\mathcal{U}^*(s,a)$) throughout the algorithm. Each time a new state-action pair is visited, max-Q, hysteretyc-Q or UCRL update their Q-functions (or mean transtition/rewards values): see for example (9), Algorithm 6 or Algorithm 3.

As mentioned before, and similarly to the Q-learning algorithm for example, this maintained uncertainty set does not implicitly solve the *exploration-exploitation tradeoff*. To solve this problem *explicitly*, we use exploration policies $(\pi_1, ..., \pi_m)$.

Let us detail the *consistency check* used in this algorithm. In this section, when a given state-action pair is tested, let us denote:

- n the total number of times (s,a) has been executed (including the latest one).
- $r'_1, ..., r'_n$ the rewards and next-states of these transitions.
- $s'_1, ..., s'_n$ the next-states of these transitions.
- $k_1, ..., k_n$ the epochs in which each these transitions happens
- $t_1, ..., t_n$ the episode stages where these transitions happen.

Algorithm 8 Pessimistic-Q algorithm

Require: $\mathcal{S}$, $\mathcal{A}$, T, δ, $\mathfrak{U}(s,a)$ for each state-action pair (s,a) , exploration policies $(\pi_1, ..., \pi_m)$
For all $(s,a) \in \mathcal{S} \times \mathcal{A}$, $t = 1, ..., T$, $\overline{r}(s,a) \leftarrow 0$, $\tilde{V}_t(s) \leftarrow 0$, $\tilde{Q}_t(s,a) \leftarrow 0$, $\mathcal{U}(s,a) \leftarrow$ smallest element in $\mathfrak{U}(s,a)$
Initialize $i \leftarrow 1$,
loop
Execute policy π_i
For each execution (s,a,r), execute the consistency check
end loop

- τ the total number of steps executed by the algorithm so far.

The null hypothesis is given by:

$$H_0 : \mathcal{U}^*(s,a) \subseteq \mathcal{U}(s,a)$$

.

The hypothesis H_0 is rejected when:

$$\sum_{j=n'+1}^{n} \left((r'_j + \tilde{V}_{t_j+1}(s'_j)) - \max_{(r,P)\in\mathcal{U}(s,a)} (r + \mathbb{E}_P \tilde{V}_{t_j+1}) \right) > 2T\sqrt{2(n-n')\log\frac{4\tau^2}{\delta}} \tag{32}$$

When H_0 is rejected, $\mathcal{U}(s,a)$ is updated to $\mathcal{U}'(s,a)$, where $\mathcal{U}'(s,a)$ is the smallest set in $\mathfrak{U}(s,a)$ such that

$$\mathcal{U}(s,a) \subseteq \mathcal{U}'(s,a)$$

and:

$$\sum_{j=n'+1}^{n} \left((r'_j + \tilde{V}_{t_j+1}(s'_j)) - \max_{(r,P)\in\mathcal{U}(s,a)} (r + \mathbb{E}_P \tilde{V}_{t_j+1}) \right) \leq T\sqrt{2(n-n')\log\frac{4\tau^2}{\delta}} \tag{33}$$

This algorithm gives the theoretical guarantee that it does not *overshoot*:

Theorem 4.1. *For any episode stage t and state-action pair (s,a),*

$$\mathcal{U}(s,a) \subseteq \mathcal{U}^*(s,a) \tag{34}$$

Proof. Firstly, for any $j = 1, ..., n$ and state-action pair (s,a), let us denote $p^{(j)}_{s,a}$ the true next-state distribution during the j-th execution of (s,a) and $r^{(j)}(s,a)$ the true expected reward for the j-th execution of (s,a).

Let us define the random variables $X_j = (r^{(j)}(s,a) + \mathbb{E}_{p^{(j)}_{s,a}} \tilde{V}_{t_j+1}) - (r'_j + \tilde{V}_{t_j+1}(s'_j))$ that form a martingale difference sequence with $\left|X_j\right| \leq T$ a.s. Using the extension of Azuma's inequality from [6] (corollary 6.9), it comes that, for any $\epsilon > 0$:

$$\mathbb{P}(\sum_{j=1}^{n} X_j > \epsilon) \leq \exp(-\frac{2\epsilon^2}{nT^2})$$

Thus, with probability at least $1 - \frac{\delta}{4\tau^2}$,

$$\sum_{j=1}^{n} \left((r^{(j)}(s,a) + \mathbb{E}_{p^{(j)}_{s,a}} \tilde{V}_{t_j+1}) - (r'_j + \tilde{V}_{t_j+1}(s'_j) \right) < T\sqrt{\frac{n}{2}\log\frac{4\tau^2}{\delta}} \tag{35}$$

Then, when the consistency check is made, with probability at least $1 - \frac{\delta}{4\tau^2}$,

$$\sum_{j=n'+11}^{n} \left((r^{(j)}(s,a) + \mathbb{E}_{p^{(j)}_{s,a}} \tilde{V}_{t_j+1}) - (r'_j + \tilde{V}_{t_j+1}(s'_j) \right) < T\sqrt{\frac{n-n'}{2}\log\frac{4\tau^2}{\delta}} \tag{36}$$

Let us now prove that $\mathcal{U}(s,a) \subseteq \mathcal{U}^*(s,a)$. Let us suppose that $\mathcal{U}^*(s,a) \subset \mathcal{U}(s,a)$. Because of (33), it comes that

$$\sum_{j=n'+1}^{n} \left((r'_j + \tilde{V}_{t_j+1}(s'_j)) - \max_{(r,P)\in\mathcal{U}^*(s,a)} (r + \mathbb{E}_P \tilde{V}_{t_j+1}) \right) > 2T\sqrt{2(n-n')\log\frac{4\tau^2}{\delta}} \tag{37}$$

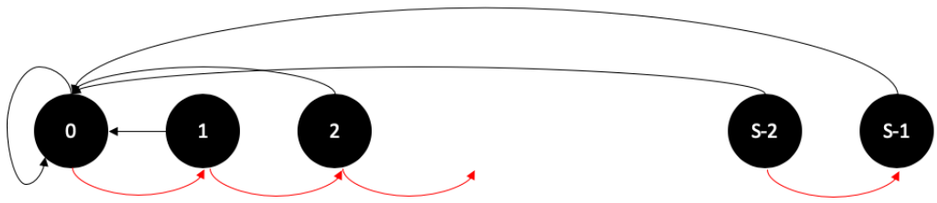

Figure 12: The chain MAMDP

otherwise, some $\mathcal{U}(s,a) \subseteq \mathcal{U}^*(s,a)$ would have been chosen instead.

However, 37 is in contradiction with .

Taking a union bound over all transitions, the total probability that (4.2.1) fails to hold is at most $\sum_{\tau=1}^{+\infty} \frac{\delta}{4\tau^2} \leq \delta$. Thus, with probability at least $1-\delta$, $\mathcal{U}(s,a) \subseteq \mathcal{U}^*(s,a)$.

□

Theorem 4.2. *For any episode stage t and state-action pair (s,a),*

$$Q_t(s,a) \leq Q_t^*(s,a) \tag{38}$$

Proof. Let us prove by induction that $\tilde{Q}_t(s,a) \leq Q_t^*(s,a)$ and $\tilde{V}_{t+1}(s) \leq V_t^*(s)$ for any time step t and state-action pair (s,a). Firstly, this holds for $t=T$ since $r(s,a) \geq \bar{r}(s,a)$. Given a state-action pair (s,a) and a time step t such that the inequality holds for any $t' > t$:

$$\begin{aligned}
\tilde{Q}_t(s,a) &= \max_{(r,P)\in\mathcal{U}(s,a)} (r(s,a) + \mathbb{E}_P \tilde{V}_{t+1}) && (39)\\
&\leq \max_{(r,P)\in\mathcal{U}^*(s,a)} (r(s,a) + \mathbb{E}_P \tilde{V}_{t+1}) && (40)\\
&\leq \max_{(r,P)\in\mathcal{U}^*(s,a)} (r(s,a) + \mathbb{E}_P V_{t+1}^*) && (41)\\
&= Q_t^*(s,a) && (42)
\end{aligned}$$

where (40) comes from Theorem 4.1 and (41) comes from the induction hypothesis.

□

4.2.2 Pessimistic-Q algorithm in a challenging MAMDP

In this section, let us introduce a multi-agent MDP designed to be challenging for multi-agent reinforcement learning algorithms (Figure 12). The dynamics are the following:

- In states $0, ..., S-2$, joint action $\mathbf{a}^*$ implies transition to next state and 0 reward.
- In state $S-1$, joint action $\mathbf{a}^*$ implies transition to the first state and 1 reward.
- In all states, any other joint action implies transition to the first state and normal reward distribution with mean $\frac{1}{2|\mathcal{S}|}$.

Contrarily to the well-mixed MAMDP, Thompson Sampling algorithm (as well as its improved versions ϵ-greedy policy exploration, ϵ-greedy action exploration) does not succeed well in this context (Figure 13), especially when the chain becomes too long.

Whereas TS only reaches a convergence rate of 90-95% in the short chain (Figure 13), the Pessimistic-Q reaches perfect result every time with the same TS exploration policy (Figure 14) on chains of the same size.

The former results show that the *Pessimistic Q-learning with exploration policies* algorithm is able to achieve better results than its exploration policies. This empirically proves the effectiveness of the conservative consistency check that avoids overshooting the true max-Q value. However, there is still no theoretical guarantee of any convergence. The next expected step of improvement for this algorithm is to find a theoretical guarantee, either on the exploration policies or on their behavior (e.g. visit optimal state-action pair often enough in a row), to ensure the convergence of the uncertainty set.

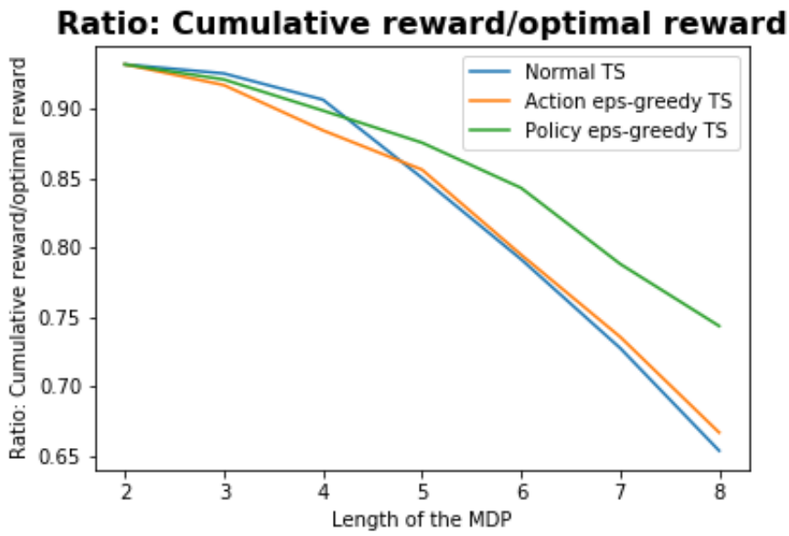

Figure 13: Thompson Sampling in the challenging chain MDP

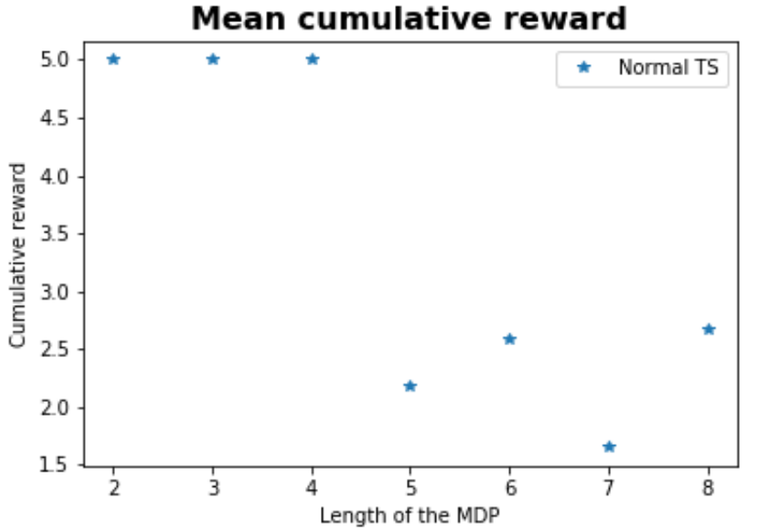

Figure 14: Pessimistic-Q with Thompson Sampling exploration policy

References

[1] Peter Auer, Nicolò Cesa-Bianchi, Yoav Freund, and Robert E. Schapire. The non-stochastic multiarmed bandit problem. *SIAM J. Comput.*, 32(1):48–77, January 2003.

[2] Thomas Jaksch, Ronald Ortner, and Peter Auer. Near-optimal regret bounds for reinforcement learning. *J. Mach. Learn. Res.*, 11:1563–1600, August 2010.

[3] Martin Lauer and Martin Riedmiller. An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In *In Proceedings of the Seventeenth International Conference on Machine Learning*, pages 535–542. Morgan Kaufmann, 2000.

[4] Shiau Hong Lim, Huan Xu, and Shie Mannor. Reinforcement learning in robust markov decision processes. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 701–709. Curran Associates, Inc., 2013.

[5] L. Matignon, G. J. Laurent, and N. L. Fort-Piat. Hysteretic q-learning : an algorithm for decentralized reinforcement learning in cooperative multi-agent teams. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 64–69, Oct 2007.

[6] Colin McDiarmid. On the method of bounded differences. In Johannes Siemons, editor, *Surveys in Combinatorics*, volume 141 of *London Mathematical Society Lecture Notes*, pages 148–188. Cambridge University Press, 1989.

[7] Francisco S. Melo. Convergence of q-learning: a simple proof.

[8] Ian Osband, Daniel Russo, and Benjamin Van Roy. (more) efficient reinforcement learning via posterior sampling, 2013.

[9] Ian Osband and Benjamin Van Roy. Why is posterior sampling better than optimism 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 2701–2710, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

[10] Emma Brunskill Rahul Sarkar. Cs 234 notes - lecture 2 making good decisions given a model of the world, March 2018.

[11] Alexander L. Strehl and Michael L. Littman. An analysis of model-based interval estimation for markov decision processes. *J. Comput. Syst. Sci.*, 74:1309–1331, 2008.