Reinforcement learning in Markov decision processes

Internship report

Apolline RODARY
L3IF - Ecole Normale Supérieure de Lyon
apolline.rodary@ens-lyon.fr

5 June 2023 - 13 July 2023
Laboratoire d'Informatique de Grenoble
Team POLARIS

Supervisors

Bruno GAUJAL
bruno.gaujal@inria.fr

Victor BOONE
victor.boone@univ-grenoble-alpes.fr

Contents

Introduction

MARKOV decision processes (MDPs) are models for single player stochastic games. In an MDP, a player can interact with the environment through actions, and observes rewards. MDPs are often used in the context of reinforcement learning (RL), where the player has no prior knowledge of the MDP and tries to maximize cumulative rewards. As such, MDPs find various applications in machine learning, game theory, healthcare, finance...

A measure of the efficiency of a learning algorithm is the *regret*. Much of the research on MDPs focuses on designing algorithms that minimize regret, and establishing interesting upper and lower bounds on the regret of these algorithms.

In this report, we lay the groundwork for the study of MDPs. After a summary of relevant definitions and notations, we start with an analysis of the offline case, where all information on the MDP is known to the player from the beginning, using the BELLMAN equation and the Value Iteration algorithm. Then, we delve into the more general problem of reinforcement learning, first with an analysis of the multi-armed bandits problem, introducing the *Optimism in the Face of Uncertainty* principle; then, with the study of general MDPs, in particular using the UCRL2 algorithm. Finally, we try to understand the shortcomings of the UCRL2 algorithm and study ameliorations such as Performance Test.

1 Definitions

1.1 MARKOV decision processes

A MARKOV decision process (MDP) is a tuple $M = \langle \mathcal{S}, \mathcal{A}, p, q \rangle$ where:

- $\mathcal{S}$ is the state space
- $\mathcal{A}$ is the action space; if $A(x)$ denotes the set of actions available from state x,

$$\mathcal{A} = \bigcup_{x \in \mathcal{S}} A(x)$$

- $p : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$[1] is the transition kernel
- $q : \mathcal{S} \times \mathcal{A} \to \Delta(\mathbb{R}_+)$ is the rewards

In such an MDP at time t, a player at state $x \in \mathcal{S}$ choosing an action $a \in A(x)$ observes a transition to a new state y with probability $p(y|x,a)$ and a random reward according to distribution $R_t \sim q(x,a)$. This defines a stochastic process $(X_t, R_t)_{t \in \mathbb{N}}$ having the MARKOV property – i.e., rewards and transitions only depend on the present state and the chosen action.

In the offline case, all information on the MDP (states, actions, reward distributions and transition probabilities for every state-action pair) is known to the player from the beginning. In the context of reinforcement learning, states and actions of the MDP are known but reward distributions and transition probabilities are not: a learning algorithm $\mathcal{L}$ – which decides how to allocate actions to every state during the play – may only depend, at any point in time, on the history of plays, state transitions and rewards observed so far.

[1] We use the notation $\Delta(X)$ for the set of probability distributions over a set X

In this report, unless stated otherwise, $\mathcal{S}$ and $\mathcal{A}$ are assumed to be finite and for all $(x, a) \in \mathcal{S} \times \mathcal{A}$ such that $a \in A(x)$, $q(x, a)$ is assumed to be of compact support, with $r(x, a) := \mathbb{E}(R(x, a))$ where $R(x, a) \sim q(x, a)$.

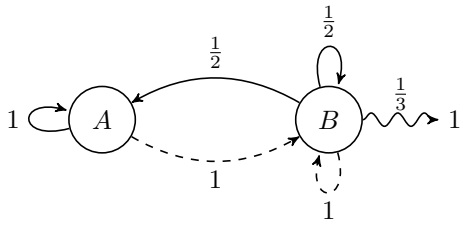


Figure 1: A two-state MDP. Solid arrows represent transition probabilities from action a, dashed arrows represent transition probabilities from action b. Rewards are represented with a wavy arrow.

In figure 1, there are two states A, B and two actions a, b. A player at state A may either play action a, staying in place with probability 1, or play action b, switching to state B with probability 1. A player at state B may either play action b, staying in place with probability 1, or play action a, giving a probability of $\frac{1}{2}$ of being in either state. All state-action pairs yield no rewards except playing action a from state B which has a $\frac{1}{3}$ chance of giving a reward of 1.

1.2 Policies

A **policy** π is a sequence of mappings $\pi_t : \mathcal{S} \to \Delta(\mathcal{A})$. A player adopting such a policy will choose, at any given t, the action $A_t \sim \pi_t(X_t)$. Any term π_t of a policy may only depend on the history $H_{t-1} = (X_1, A_1, \ldots, X_{t-1}, A_{t-1})$.

For an MDP M, a policy π and an initial state x, if V is a random variable that depends on the history of play on M using policy π, we write

$$\mathbb{E}_x^{M,\pi}(V) := \mathbb{E}(V(M, x, \pi))$$

Deterministic stationary policies are written as $\pi : \mathcal{S} \to \mathcal{A}$. Unless specified otherwise, policies in this report are assumed to be deterministic and stationary. More complex policies – which may depend on the history of transitions and rewards – are identified as **learning algorithms** and are written as $\mathcal{L}$ instead.

For a stationary deterministic policy π, we write $P^\pi \in M_{|\mathcal{S}|}(\mathbb{R})$ the transition probability matrix induced by π, i.e. the stochastic matrix defined by $P^\pi_{x,y} = p(y \,|\, x, \pi(x))$. We also write $r^\pi \in \mathbb{R}_+^{|\mathcal{S}|}$ the vector defined by $r^\pi(x) = r(x, \pi(x))$. The Markov chain induced by π is the Markov chain over state space $\mathcal{S}$ of transition probability matrix P^π.

1.3 Finite horizon, discounted, average

The general problem is how to choose actions so to maximize total rewards on average. There are three flavours to this problem:

Finite horizon. A finite number of steps T are taken; the benchmark for a policy π is

$$\mathbb{E}_x^\pi \left(\sum_{t=1}^{T} R_t \right)$$

Discounted. We consider a *discount factor* $\gamma \in (0, 1)$. The benchmark for a policy π is

$$\mathbb{E}_x^\pi \left(\sum_{t=1}^{+\infty} \gamma^{t-1} R_t \right)$$

It is guaranteed that the sum does indeed converge if the state space and the action space are finite and the reward distributions for all state-action pairs are compact.

Average/undiscounted. An infinite number of steps are taken. The benchmark for a policy π is

$$\lim_{T \to \infty} \frac{1}{T} \mathbb{E}_x^\pi \left(\sum_{t=1}^{T} R_t \right)$$

1.4 Gain

The **gain** of a stationary policy π is the quantity that we are trying to maximize in the average case:

$$g^\pi(x, M) := \lim_{T \to \infty} \frac{1}{T} \mathbb{E}_x^\pi \left(\sum_{t=1}^{T} R_t \right)$$

For finite state and action spaces, this limit is guaranteed to exist for any stationary policy. Otherwise, we consider the lim sup.

A policy is said to be *gain-optimal* or simply *optimal* if no policy achieves a strictly greater gain.

For example, in figure 1, starting at state A, the only policy with gain greater than zero is to play action a from state A and action b from state B. The gain of that policy is $\frac{2}{9}$, because on average the player is at state B two thirds of the time and gains a reward of 1 one third of the time playing from state B.

1.5 Communicating MDPs

Hypotheses on the structure of an MDP, analagous to common hypotheses on Markov chains – are often made when studying RL algorithms [3]:

Communicating. An MDP is said to be communicating if for any given pair of states $(x, y) \in \mathcal{S}^2$, there exists a policy π such that state y can be reached from state x using policy π in finitely many steps with nonzero probability, i.e. there exists $k \in \mathbb{N}$ such that

$$(P^\pi)_{x,y}^k > 0$$

In a communicating MDP, the gain of a policy does not depend on the initial state and so it is written as $g^\pi(M)$ or simply g^π [7].

Weakly communicating. An MDP is said to be weakly communicating if the state space can be partitioned into two subsets $\mathcal{S}^C$, $\mathcal{S}^T$ such that

- For any given pair of states $(x, y) \in \mathcal{S}^C$, there exists a policy π such that state y can be reached from state x using policy π in finitely many steps with nonzero probability.
- For any state $x \in \mathcal{S}^T$, x is transient under all deterministic stationary policies

Ergodic. An MDP is said to be ergodic if all states are recurrent under all policies, i.e. the Markov chain induced by any policy is recurrent.

1.6 Invariant measure

In an ergodic, aperiodic MDP, under a fixed policy π, a single stationary distribution over states exists. It is written as μ^π; informally, the probability of presence at any state x while iterating the policy approaches $\mu^\pi(x)$.

It follows that in an ergodic, aperiodic MDP, the gain can be expressed as $g^\pi = \langle \mu^\pi \,|\, r^\pi \rangle$.

1.7 Bias

Under a given policy, the **bias** or **potential** of a state is defined as follows:

$$h^\pi(x) := \lim_{T \to \infty} \mathbb{E}_x^\pi \left(T g^\pi - \sum_{t=1}^{T} R_t \right)$$

The limit does indeed exist for any ergodic MDP. Otherwise, the Cesàro sum may be considered instead.

Informally, the rewards are expected to be equivalent to Tg^π as T grows large. However, a deviation specific to the initial state is observed. In figure 2 for example, the policy π that always plays action b has a gain of 1 and a bias $h^\pi(A) = -1$, $h^\pi(B) = 0$.

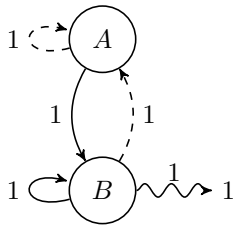


Figure 2: A deterministic two state MDP with action a (dashed) and b (solid)

Among policies achieving optimal gain, the one that maximizes bias is called bias-optimal. From here on, quantities relating to this policy are written with an asterisk.

1.8 Regret

The regret of a learning algorithm $\mathcal{L}$ after T steps is the difference between the expected rewards obtained by iterating $\mathcal{L}$ and the expected rewards of an optimal policy[2]. Writing g^* as the gain of an optimal policy, i.e. a policy π^* such that $\forall \pi \ g^{\pi^*} \geqslant g^{\pi}$, the regret can be expressed as

$$\mathrm{Reg}(T) = Tg^* - \mathbb{E}^{\mathcal{L}}\left(\sum_{t=1}^{T} R_t\right)$$

No-regret algorithms are learning algorithms with sublinear regret:

$$\mathrm{Reg}(T) \underset{T\to+\infty}{=} o(T)$$

2 Offline MDPs

In the offline case, all information on the MDP is known to the player before any play is done.

2.1 Finite horizon

The finite horizon problem can be solved in the offline case with dynamic programming.

Let us consider an MDP $\langle \mathcal{S}, \mathcal{A}, p, q \rangle$. We want to maximize the expected total rewards after a fixed number of steps T starting at a given state x_0. For any $(x,t) \in \mathcal{S} \times \{0,\dots,T\}$, we write $V_t^*(x)$ the expected rewards obtained on the next t steps assuming that we are starting at position x and playing an optimal policy. Trivially, for any state x,

$$\begin{aligned} V_0^*(x) &= 0 \\ V_1^*(x) &= \max_{a\in A(x)} r(x,a) \end{aligned}$$

This generalizes to the following recursive definition: for all $x \in \mathcal{S}$, for all $t \in \{0,\dots,T-1\}$,

$$V_{t+1}^*(x) = \max_{a\in A(x)} \underbrace{\left(r(x,a) + \sum_{y\in\mathcal{S}} p\left(y \,|\, x,a\right) \times V_t^*(y)\right)}_{:=Q(x,a)}$$

In other words, the best action that can be taken from state x when there are $t+1$ steps left is the one that maximises the expected instant rewards $r(x,a)$ added to the expected rewards for the next t steps assuming the best actions will also be taken. This is referred to as the *finite horizon Bellman equation*.

This naturally leads to the following algorithm:

[2]What is called "regret" in this report is sometimes referred to as "expected regret" in the literature, where "regret" is instead defined as the random variable $Tg^* - \sum_{t=1}^{T} R_t$. Instead, the latter is referred to as "observed regret" in this report.

Algorithm 1: Solving the finite horizon MDP problem in the offline case

```
for x ∈ S do
    V_0^*(x) ← 0
end
for t ← 1 to T do
    for x ∈ S do
        Compute Q_{t−1}(x, a) for all a ∈ A(x)
        V_t^*(x) ← max_{a∈A(x)} Q(x, a)
        B(x, t) ← argmax_{a∈A(x)} Q(x, a)
    end
end
for t ← T to 1 do
    Observe current state x
    Play B(x, t)
end
```

2.2 Value iteration

Let us write the previous equation in matrix form:

$$V_{t+1}^* = \max_{\pi \in \Pi} (r^\pi + P^\pi \times V_t^*)$$

where Π is the set of all deterministic stationary policies. Noting that the value of $(r^\pi + P^\pi \times V_t^*)(x)$ for a given state x only depends on the action chosen for x in policy π, the above maximum is well defined as there is a policy that maximizes this quantity for all x.

The function $V \mapsto \max_\pi(r^\pi + P^\pi V)$ is called the **Bellman operator**. For an aperiodic, ergodic MDP, recursive application of the Bellman operator consists in adding a vector that eventually converges towards a real constant, the gain of an optimal policy:

$$V_{t+1}^* - V_t^* \underset{t \to +\infty}{=} g^* + o(1)$$

To understand this result, consider the Markov chain induced by any policy. Because the MDP is ergodic and aperiodic, the Markov chain is irreducible and aperiodic, and so a single stationary distribution μ^π exists. After many plays, the probability of presence on any given state x approaches a constant – $\mu^\pi(x)$ – which does not depend on the starting state; the expected reward approaches $\langle \mu^\pi \,|\, r^\pi \rangle$ which is the gain of policy π.

In the average case, a popular algorithm for finding a near-optimal policy is *Value iteration* [7]. Value iteration can be seen as a generalization of the previous algorithm, which stops when further application of the Bellman operator seems to converge to addition of a constant. Specifically, when the difference between two consecutive applications of the Bellman operator is a vector with a span[3] less than some ε, the algorithm stops and returns the last policy, which is guaranteed to be 2ε-gain-optimal, meaning no policy can achieve a gain greater than $2\varepsilon + g^\pi$.

[3]The *span* of a vector is a measure of how far apart components of that vector can be: $span(V) := \max_{i,j} |V_i - V_j|$. This is unrelated to linear span.

Algorithm 2: Value iteration

$V_0 \leftarrow (0, \dots, 0)$
$t \leftarrow 0$
repeat
  for $x \in \mathcal{S}$ **do**
    Compute $Q_t(x, a)$ for all $a \in A(x)$
    $V_{t+1}^*(x) \leftarrow \max_{a \in A(x)} Q_t(x, a)$
    $\pi(x) \leftarrow \operatorname{argmax}_{a \in A(x)} Q_t(x, a)$
  end
  if $span(V_{t+1}^* - V_t^*) < \varepsilon$ **then**
    return π
  end
  $t \leftarrow t + 1$

Theorem. *When the Value iteration algorithm halts, the returned policy π is 2ε-gain-optimal.*

The halting of the value iteration algorithm was not studied during the internship. However, a proof of the near-optimality of the returned policy if and when it does stop is provided in the appendix.

Note that the value iteration algorithm not only provides a near-optimal policy, but also its gain with precision ε – that is, any component of the vector $V_{t+1}^* - V_t^*$ when the algorithm ends – and the bias vector $h = V_t^* - tg^*$.

2.3 Getting the invariant measure of a policy

Once a policy has been generated, it may be interesting to estimate its invariant measure to study the long-term behaviour of plays on the MDP. A fast and precise approach is to use the value iteration algorithm on modified MDPs with restricted action spaces.

Consider an MDP $\langle \mathcal{S}, \mathcal{A}, p, q \rangle$ and any state $x \in \mathcal{S}$. The limit probability of presence at state x, $\mu^\pi(x)$ can be obtained by getting the gain of the same policy π on the following MDP M'_x:

- The state space and the transition kernel of the initial MDP are unchanged
- From every state y, the only possible action is $\pi(y)$
- All rewards are 0, except the reward for the state-action pair $(x, \pi(x))$ which is 1

Proof. Trivially, every run of value iteration on a modified MDP with action space A' will return the initial policy π, as for every state there is only one action to choose. The gain of that policy on any modified MDP M'_x is $g = \langle \mu^\pi \,|\, r^\pi \rangle$, where r^π is defined as $\mathbf{1}_{(x,\pi(x))}$ and so the dot product is simply $\mu^\pi(x)$. □

Though sampling the MDP directly is easily done, this approach is more efficient: indeed, in the ergodic case, Value iteration has geometric convergence, as a result of the geometric convergence of presence probability distributions in ergodic Markov chains (see Theorem 4.9 in [6]). Meanwhile, sampling has a standard error proportional to $\frac{1}{\sqrt{T}}$.

Algorithm 3: Computing the invariant measure μ^π of a policy π

Input: An MDP $M = \langle \mathcal{S}, \mathcal{A}, p, q \rangle$

forall $x \in \mathcal{S}$ **do**
  $A'(x) \leftarrow \{\pi(x)\}$
end
forall $x \in \mathcal{S}$ **do**
  $q'_x \leftarrow \mathbf{1}_{(x,\pi(x))}$
  $M'_x \leftarrow \langle \mathcal{S}, \mathcal{A}', p, q'_x \rangle$
  Get gain g from ValueIteration(M')
  $\mu^\pi(x) \leftarrow g$
end

3 Multi-armed bandits

The *Multi-armed bandit problem* is a simple example of an MDP learning optimization problem, where the studied MDP is stateless and no transition is observed. The problem can be formulated as such:

There are K slot machines, all with unknown and presumably distinct reward distributions on $[0, 1]$. For $i \in \{1, \dots, K\}$, we write $r(i)$ the expected reward of playing on machine i, and r^* the reward of an optimal machine:

$$r^* = \max_{i \in \{1,\dots,K\}} r(i)$$

The regret of choosing machine i at any point in time is written as $\Delta_i := r^* - r(i)$.

Assuming we have infinitely many tokens to play on any of the machines, how can we choose which slot machine to play on at any point in time so to maximize average rewards?

3.1 Explore then commit

An intuitive approach to the multi-armed bandits problem is the **explore-then-commit** algorithm. In this algorithm, all machines are played a certain number of times and an estimate of the average rewards of each machine is computed. Then, the machine with the greatest empirical average is always played from then on.

Though the probability of having total rewards not equivalent to optimal rewards can be made arbitrarily low, there is always a chance that a suboptimal machine is assumed to be optimal, which causes regret to grow linearly. This highlights the *exploration vs exploitation dilemma*: though learning algorithms need to exploit strategies that seem to maximize rewards as much as possible, they also need to explore suboptimal strategies arbitrarily many times to ensure that the best strategy has been found.

3.2 UCB and the OFU principle

In the UCB algorithm [1], all machines are initially played on exactly once, creating a first estimate of the rewards of every machine; then, the machine that is played at any point in time is the one that maximizes the upper bound of the constructed confidence interval for that machine. Confidence

intervals are constructed to grow over time so that all machines, even those that are deemed suboptimal, are played on arbitrarily many times, giving higher priority to machines that have higher uncertainty but promising upper bounds.

UCB is an example of algorithm that follows the *"Optimism in the Face of Uncertainty"* (OFU) principle – it is often beneficial to be optimistic about the potential outcomes of an action.

Algorithm 4: UCB

for $i \leftarrow 1$ **to** K **do**
  Play machine i
end
repeat
  Play machine i that maximizes

$$\tilde{r}(i) + \sqrt{\frac{2 \log t}{N(i)}}$$

  where

- $\tilde{r}(i)$ is the average observed reward on machine i
- t is the current time
- $N(i)$ is the number of plays on machine i so far

Theorem. *After T iterations of UCB,*

$$\mathrm{Reg}(T) \leqslant \left(8 \sum_{i : r(i) < r^*} \left(\frac{\log T}{\Delta_i} \right) + \left(1 + \frac{\pi^2}{3} \right) \left(\sum_{i=1}^{K} \Delta_i \right) \right)$$

This can be proven by first establishing that the expected number of plays on any suboptimal machine i after T total plays is no more than $\left\lceil \frac{8 \log T}{\Delta_i^2} \right\rceil + \frac{\pi^2}{3}$, which itself can be done using union bound and the Chernoff-Hoeffding inequality. The result is established and proven in [1]; a refined version of this proof is given as an appendix.

Simulations from random instances of the multi-armed bandits problem does seem to show logarithmic regret (see figures 3, 4), however the upper bound given by the previous theorem is usually much larger than the experimental results.

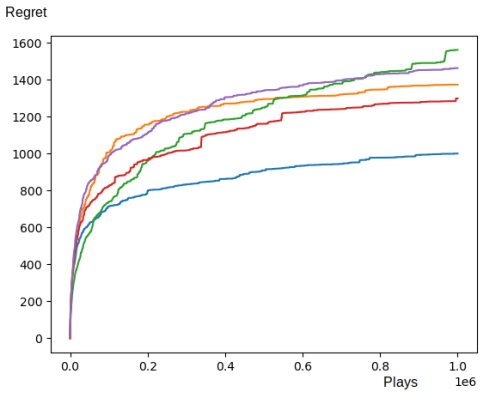
Another interesting upper bound on the regret obtained by UCB after T iterations is

$$\mathrm{Reg}(T) \leqslant \sqrt{32 \left| \mathcal{A} \right| T \log T}$$

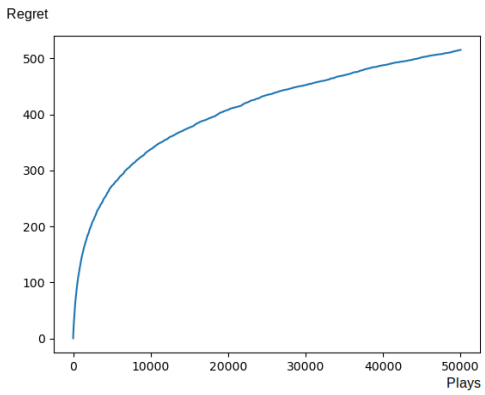
which can be proven using the Azuma-Hoeffding inequality, the Cauchy-Schwarz inequality, and upper bounding sums with integrals.

4 Reinforcement learning in general MDPs

The previous section has established the efficiency of the OFU principle in the case of the multi-armed bandits problem, with algorithms such as UCB. In the broader scope of MDPs, how such an



(a) 5 different runs of UCB, 1000000 plays.



(b) Averaging 30 different runs of UCB, 50000 plays.

Figure 3: Observed regrets ($\sum_{t=1}^{T}(r^* - R_t)$) during runs of UCB on random instances of the multi-armed bandits. Rewards of each machine are BERNOULLI variates with probability of rewards drawn uniformly in $[0, 1]$

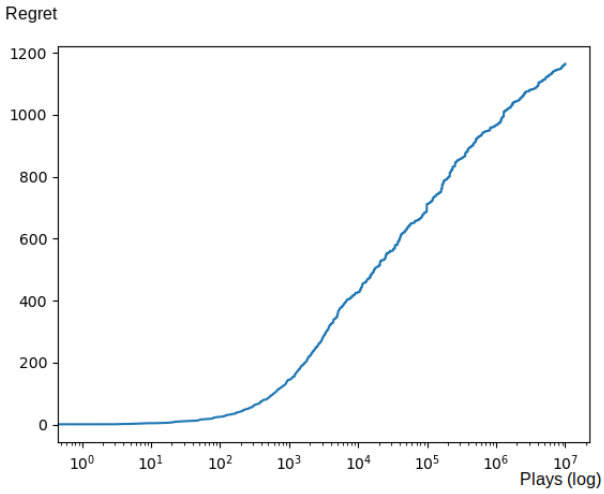


Figure 4: Observed regrets (on a log scale) during a run of UCB on a random instance of the multi-armed bandits, 10^7 plays

approach can be generalized is nontrivial. An algorithm on which much of the more recent literature on MDPs is based is UCRL-2 [5], which – for an ergodic MDP – builds a model of the MDP based on estimates for every transition chance and reward distribution, and consistently applies policies that are optimal for a plausible "optimist MDP" $\hat{M}$ during episodes that grow exponentially long.

4.1 UCRL-2

In episodic algorithms, policies are consistently applied throughout episodes of arbitrarily length, until a stopping condition is reached and the algorithm switches to a new policy. In UCRL-2 in particular, at the beginning of every episode, a confidence set of plausible models for the true MDP M – containing the true MDP with high probability – is built and denoted as $\tilde{\mathcal{M}}$; a policy is chosen which maximizes the best possible gain in $\tilde{\mathcal{M}}$, i.e. achieves gain $\tilde{g}^* = \max_\pi \sup_{\tilde{M}\in\tilde{\mathcal{M}}} g(\pi, \tilde{M})$. The episode ends when a state-action pair has been observed as many times during the episode as it has been throughout all previous episodes (*Doubling trick*).

In the below description of the UCRL-2 algorithm, we write the number of visits of a state-action pair (x, a) by time t as $N_t(x, a)$, and t_k refers to the time at the beginning of episode k. Rewards are still considered within $[0, 1]$.

Algorithm 5: UCRL-2

Input: A parameter δ

$t \leftarrow 1$
$k \leftarrow 0$
repeat
  $k \leftarrow k + 1$
  $t_k \leftarrow t$
  Define confidence set $\tilde{\mathcal{M}}$
  $\pi \leftarrow \text{argmax}_\pi \sup_{\tilde{M}\in\tilde{\mathcal{M}}} g(\pi, \tilde{M})$
  repeat
    Observe state x
    $a \leftarrow \pi(x)$
    Play a
    $t \leftarrow t + 1$
  until $N_t(x, a) = \max(1, 2N_{t_k}(x, a))$;

$\tilde{\mathcal{M}}$ is defined as the set of MDPs $\langle \mathcal{S}, \mathcal{A}, p, q\rangle$ with

$$\forall x \in \mathcal{S} \quad \forall a \in A(x) \quad \begin{cases} \|p(\cdot\,|\,x, a) - \tilde{p}(\cdot\,|\,x, a)\|_1 < d(x, a) \\ |r(x, a) - \tilde{r}(x, a)| < d'(x, a) \end{cases}$$

where $\tilde{p}, \tilde{r}$ are the estimates for the transition probabilities and average rewards of every state-action pair, and d, d' are defined as follows:

$$d(x, a) = \sqrt{\frac{7\log(2|\mathcal{S}||\mathcal{A}|\frac{t}{\delta})}{2\max(1, N_t(x, a))}}$$

$$d'(x, a) = \sqrt{\frac{14|\mathcal{S}|\log(2|\mathcal{A}|\frac{t}{\delta})}{\max(1, N_t(x, a))}}$$

Finding the optimistic MDP $\hat{M} \in \tilde{\mathcal{M}}$ for which maximal gain can be achieved is done with an algorithm called *extended value iteration* which is detailed below.

4.2 Extended value iteration

In value iteration, we try to find a fixed point of the Bellman operator, which gives us the best policy to choose over a finite state space and a finite action space. In extended value iteration (EVI), we are now trying to optimize the following:

$$\forall x \in \mathcal{S} \quad h(x) = \max_{a \in A(x)} \left(r(x,a) + \sum_{y \in \mathcal{S}} p\left(y \,|\, x,a\right) h(y) \right)$$

for p, r satisfying the previously established conditions.

The space of plausible MDPs is formalized in [4] as an *extended MDP* or *bounded-parameter MDP* with an *extended action set* where every plausible transition probability distribution is considered as a separate action; extended value iteration, much like the standard finite-horizon value iteration, then searches a fixed point of the Bellman operator as it applies for the considered bounded-parameter MDP.

In the optimist MDP, all rewards can be assumed to be as big as possible:

$$\forall (x,a) \in \mathcal{S} \times \mathcal{A} \quad r(x,a) = \tilde{r}(x,a) + d'(x,a)$$

and maximizing $\sum_{y \in \mathcal{S}} p\left(y \,|\, x,a\right) V_t(x)$ over all plausible transition probabilities for $p\left(\cdot \,|\, x,a\right)$ is a linear optimization problem:

Maximize $\langle p\left(\cdot \,|\, x,a\right) \,|\, V_t \rangle$ under the constraints:

$$\begin{aligned} &\left\| p\left(\cdot \,|\, x,a\right) - \tilde{p}\left(\cdot \,|\, x,a\right) \right\|_1 < d(x,a) \\ &\sum_{y \in \mathcal{S}} p\left(y \,|\, x,a\right) = 1 \end{aligned}$$

The extended value iteration algorithm can then be written simply as such [5]:

Algorithm 6: Linear optimization algorithm for extended value iteration

Input: A probability vector $\tilde{p}$ over $\mathcal{S}$, a distance d, a bias vector h

Sort states descendingly according to their h-values ($h(x_1) \geqslant \ldots \geqslant h(x_n)$)
$p \leftarrow \tilde{p}$, $i \leftarrow 1$, $j \leftarrow n$
while $i < j$ **do**
    Decrease $p(x_j)$ and increase $p(x_i)$ as much as possible:
    $m \leftarrow \min(\frac{d}{2}, 1 - p(x_i), p(x_j))$
    $p(x_i) \leftarrow p(x_i) + m$
    $p(x_j) \leftarrow p(x_j) - m$
    $d \leftarrow d - 2m$
    if $m = \frac{d}{2}$ **then**
        break
    end
    if $m = 1 - p(x_i)$ **then**
        $i \leftarrow i + 1$
    end
    if $m = p(x_j)$ **then**
        $i \leftarrow i - 1$
    end
end

Algorithm 7: Extended value iteration

$V_0 \leftarrow (0, \ldots, 0)$
$t \leftarrow 0$
repeat
    for $x \in \mathcal{S}$ **do**
        $$V_{t+1}^*(x) \leftarrow \max_{a \in A(x)} \left(\hat{r}(x,a) + \max_{p(\cdot\,|\,x,a) \approx \hat{p}(\cdot\,|\,x,a)} \langle p(\cdot\,|\,x,a) \,|\, V_t \rangle \right)$$
        $$\pi(x) \leftarrow \operatorname*{argmax}_{a \in A(x)} \textit{of the above}$$
        where $\hat{r}(x,a) = \tilde{r}(x,a) + d'(x,a)$ and $p(\cdot\,|\,x,a) \approx \hat{p}(\cdot\,|\,x,a)$ means within distance $d(x,a)$ for $\|\cdot\|_1$, and the inner maximum is computed with algorithm 6 and inputs $p(\cdot\,|\,x,a)$, $d(x,a)$ and V_t.
    end
    if $span(V_{t+1}^* - V_t^*) < \varepsilon$ **then**
        return π
    end
    $t \leftarrow t + 1$

Much like in value iteration, we get to stop once we reach a policy that achieves a gain arbitrarily close to the optimal gain (for the model's optimist MDP $\hat{M}$). In UCRL-2, we set $\varepsilon = \frac{1}{\sqrt{t}}$.

4.3 Regret graph and Bellman gap

In the context of this internship, UCRL-2 was mostly tested against a Riverswim MDP (see figure 5). In Riverswim, the player may either choose action RIGHT – giving a big chance of staying in the same state, a small chance of progressing one state to the right, and a smaller chance of slipping back to the left – or LEFT, moving back for sure to a lower state. Low rewards may be obtained by choosing to go left while in state 1, and higher rewards may be obtained by choosing to go right while in state n. As such, Riverswim is "hard to learn" in that the greedy approach of going left and achieving a nonzero reward may seem like the best approach until the MDP has been fully explored and the very last state has been reached, obtaining no rewards in the process; yet the only optimal policy is to always go right, which has a gain of approximately 0.77.

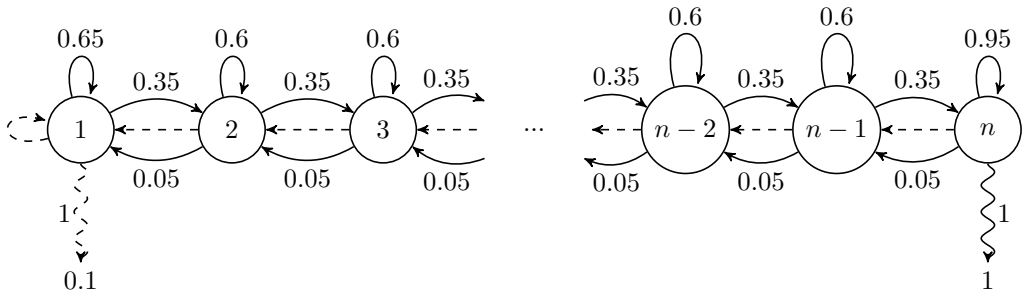


Figure 5: The n-state riverswim MDP [8]. The player starts at state 1 and may choose action RIGHT (solid arrows) or LEFT (dashed arrows) from every state.

In figure 6, we have plotted (in blue) observed regrets on an 8-state Riverswim MDP over the course of 1000000 iterations of UCRL-2.

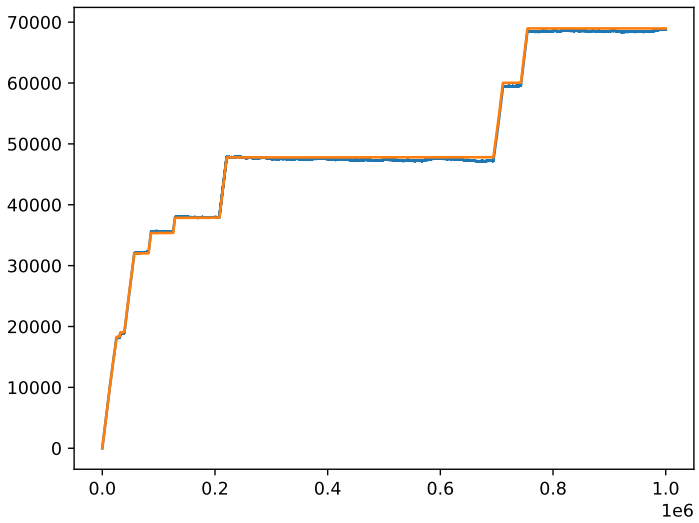


Figure 6: Observed regrets (blue) and gap regrets (orange) of a UCRL-2 run on the 8-state Riverswim

We can easily discriminate episodes exploring suboptimal policies (where the observed total regrets spikes up) and those exploring an optimal policy (where the observed total regrets seem to plateau). As expected, new exploration episodes keep happening but the optimal policy – going always right – is still exploited for episodes that grow longer and longer.

Because the observed rewards $(\sum_{t=1}^{T}(g^* - R_t))$ are inherently stochastic measure, there are unwanted fluctuations in the graph and sometimes the observed regret seems to decrease over time as lucky plays score strictly better than the optimal strategy. A better benchmark for the quality of a play (x, a) is the Bellman gap:

$$\Delta^*(x,a) = g^* - r(x,a) + h^*(x) - \sum_y p\,(y\,|\,x,a)\,h^*(y)$$

Informally, the Bellman gap measures how far away from the optimal gain we can hope to be by playing a from state x, accounting for a difference in bias induced by the random transition that is then observed. Writing the Bellman gap definition as a matrix equation, we get

$$\Delta^*(\pi) = g^* - r^\pi + (I - P^\pi)h^*$$

which is guaranteed to be greater or equal to 0, and only equal to 0 if the policy is optimal. In fact, for $\pi = \pi^*$, the above equation can be rewritten as

$$\Delta * (\pi) = g^\pi + h^\pi - r^\pi - P^\pi h^\pi$$

and we have previously established that

$$g^\pi + h^\pi = r^\pi + P^\pi h^\pi$$

el Bellman gaps over the course of a UCRL-2 run are still stochastic in that they depend of the state of the MDP at any point in time and of the policies that are chosen, which themselves still depend on the history of observed transitions and rewards. Therefore, they are still subject to noise and deviations from the true, average regret. However, gap regret graphs are at least (weakly) increasing and have a consistent slope throughout any given episode (see figure 6). In particular, gap regret does not increase during episodes where an optimal policy has been adopted.

4.4 Shortcomings of UCRL-2

Though UCRL-2 – as well as its many variants – is a powerful algorithm to force exploration in MDPs and to minimize regret asymptotically, it inevitably adopts suboptimal policies infinitely many times for episodes that grow arbitrarly long. This motivates the creation of algorithms that seem to limit to a maximum the duration or frequency of bad episodes, while still permitting exploration. Such an algorithm UCRL-PT or UCRL with Performance Test [2]:

Algorithm 8: UCRL-PT

Input: A parameter α

$t \leftarrow 1$
$k \leftarrow 0$
repeat
  $k \leftarrow k + 1$
  $t_k \leftarrow t$
  Define confidence set $\tilde{\mathcal{M}}$
  $\pi \leftarrow \text{argmax}_\pi \sup_{\tilde{M} \in \tilde{\mathcal{M}}} g(\pi, \tilde{M})$ (EVI)
  repeat
    Observe state x
    $a \leftarrow \pi(x)$
    Play a
    $t \leftarrow t + 1$
  until $N_t(x,a) = \max(1, 2N_{t_k}(x,a))$ *or* $\tilde{g}_t(\pi) \leqslant \tilde{g}_t^* - \sqrt{\alpha \frac{\log t}{t}}$;

It can be proven that a no-regret algorithm must use suboptimal policies infinitely often; practically, what UCRL-PT does is increase the frequency of bad episodes but limit their duration.

To study the behaviour of UCRL-PT during bad episodes, I was tasked to compare the evolution of the gain of an optimal policy on the optimist model $\hat{M} \in \tilde{\mathcal{M}}$ (by repeatedly running EVI) and the evolution of the gain of the episode's policy π on the optimist MDP (by repeatedly running EVI but restraining the actions to those of π) during every play of a bad episode. The two quantities are expected to be initially equivalent as the policy chosen for an episode is the one that is optimal for the optimist MDP at the beginning of the episode. The motivation is that when both quantities branch off, EVI would start to predict that the current policy is suboptimal and so this could give insight as to when UCRL-PT interrupts the current policy.

On long runs of UCRL2 on an 8-state Riverswim MDP, finding a bad episode of duration 1000 or more after the optimal policy has already been adopted, the evolution of gains on the 1000 first plays – described in figure 7 – show that EVI seems to lock onto the bad policy for a few hundred plays, at which point the bad policy is observed to perform badly.

Note that the gain estimates barely evolve (from 0.90642 to 0.90630 in figure 7) as these are

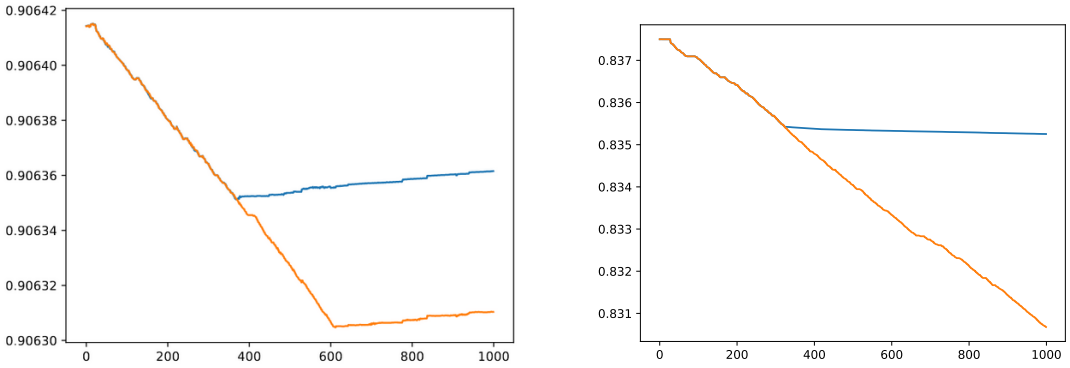


Figure 7: 1000 plays of a bad policy during Riverswim. In orange, EVI estimates of gain on the optimist model $\hat{M}$ for the bad policy. In blue, EVI estimates of the optimal gain on the optimist model $\hat{M}$.

built on all previous plays; because the bad episodes that are studied here are deliberately chosen to appear late during the play (after hundreds of thousands of iterations for the figure on the right, and millions of iterations for the figure on the left), the 1000 play window that is observed is comparatively small next to the history up until the beginning of the episode.

Finally, I was asked to study in further detail the shortcomings of UCRL-2 by constructing ergodic MDPs that UCRL-2 wouldn't perform well on, ideally highlighting how the optimist MDPs constructed during the play would be unrepresentative of the structure of the true MDP. This is somewhat apparent in Riverswim, where although a state i may only lead to states $i-1$, i or $i+1$ in a single play, it is assumed at all times that there are non-zero transition probabilities from any state to any other state. Unfortunately, this came near the very end of the internship and no interesting results were identified.

Conclusion

OFU algorithms such as UCB for the multi-armed bandits and UCRL-2 for the more general MDPs prove to be powerful algorithms for minimizing regret in an RL setting and responding efficiently to the exploitation vs exploration dilemma. Nonetheless, UCRL-2 remains unsatisfactory as it forces the exploration of bad policies for arbitrarily long episodes, arbitrarily many times. In practice, limiting the duration of bad episodes when they are observed to be bad enough is preferable; we have shown experimentally that late enough in a bad episode, running EVI suddenly deviates from the policy adopted at the beginning of the episode, which could be used as a trigger for interrupting an episode early.

My contribution

Throughout the internship, the objective was to delve into the extensive body of work surrounding MDPs and to get a general understanding of the subject as a whole. The already existing literature on MDPs being quite dense, it was naturally difficult to grasp the entire scope of the field and make notable contributions. As a result, my focus mostly revolved around understanding, implementing and testing practical algorithms such as Value Iteration, UCB and UCRL-2, as well as understanding and reestablishing some well-known properties of MDPs and important results of these algorithms, as presented in this report.

The programming part of this internship focused on:

- Representing MDPs, policies, and simulating transitions and rewards
- Implementing the value iteration algorithm, as well as a helper function for getting the invariant measure of a policy
- Implementing the UCB algorithm for the multi-armed bandits
- Implementing the UCRL-2 algorithm (including extended value iteration), testing it against Riverswim
- Building regret graphs and gap regret graphs
- Analyzing the behaviour of extended value iteration throughout a suboptimal episode during UCRL-2 runs

Because many of these algorithms can be computationally expensive, I have chosen to implement all of the above in C++, except for the UCB algorithm which I have implemented in Python. Additionally, my code on MDPs was exposed to a Python library for easier simulations, via swig (which proved to be more time-consuming than originally expected).

Most of the programming work done during the internship is made available as a repository on my GitHub account.

Beyond the programming part, much of the internship was dedicated to reading and understanding proofs surrounding the implemented algorithms, in particular UCB and UCRL2.

References

[1] P. Auer, N. Cesa-Bianchi, and P. Fischer. "Finite-time Analysis of the Multiarmed Bandit Problem". In: *Machine Learning* 47.2-3 (2002), pp. 235–256. DOI: http://doi.org/10.1023/A:1013689704352.

[2] Victor Boone and Bruno Gaujal. "The Regret of Exploration and the Control of Bad Episodes in Reinforcement Learning". In: *Proceedings of the 40th International Conference on Machine Learning (ICML 2023)*. Hawaii-Honolulu, United States, 2023, pp. 2824–2856. URL: https://inria.hal.science/hal-04161584.

[3] Ronan Fruit. "Exploration-exploitation dilemma in Reinforcement Learning under various form of prior knowledge". PhD thesis. Université de Lille 1, Sciences et Technologies; CRIStAL UMR 9189, 2019. URL: https://theses.hal.science/tel-02388395.

[4] Robert Givan, Sonia Leach, and Thomas Dean. "Bounded-parameter Markov decision processes". In: *Artificial Intelligence* 122.1 (2000), pp. 71–109. ISSN: 0004-3702. DOI: https://doi.org/10.1016/S0004-3702(00)00047-3. URL: https://www.sciencedirect.com/science/article/pii/S0004370200000473.

[5] Thomas Jaksch, Ronald Ortner, and Peter Auer. "Near-optimal Regret Bounds for Reinforcement Learning". In: *Journal of Machine Learning Research* 11.51 (2010), pp. 1563–1600. URL: http://jmlr.org/papers/v11/jaksch10a.html.

[6] David A. Levin, Yuval Peres, and Elizabeth L. Wilmer. *Markov Chains and Mixing Times, Second Edition*. 2017. ISBN: 978-1-4704-2962-1.

[7] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Series in Probability and Statistics. 1994. DOI: 10.1002/9780470316887.

[8] Alexander L. Strehl and Michael L. Littman. "An analysis of model-based Interval Estimation for Markov Decision Processes". In: *Journal of Computer and System Sciences* 74.8 (2008). Learning Theory 2005, pp. 1309–1331. ISSN: 0022-0000. DOI: https://doi.org/10.1016/j.jcss.2007.08.009. URL: https://www.sciencedirect.com/science/article/pii/S0022000008000767.

A Proofs

Near-optimality of the policy returned by value iteration

Proof. It is not proven here that the algorithm does eventually stop; we are only trying to verify that when it does, the returned policy achieves a gain greater than $g^{\pi'} - 2\varepsilon$ for any policy π'.

By definition,

$$g^\pi + h^\pi = r^\pi + P^\pi h^\pi$$

By construction, at the time $t+1$ of halting,

$$V_{t+1} = r^\pi + P^\pi V_t$$

Therefore,

$$\begin{aligned} r^\pi &= (I - P^\pi)V_t + V_{t+1} - V_t \\ &> \tilde{g} - \varepsilon + (I - P^\pi)V_t \end{aligned}$$

where $\tilde{g}$ is any value in vector $V_{t+1} - V_t$ (presumed to be close to the real gain g).

For any $l \in \mathbb{N}$, $(P^\pi)^l$ is a stochastic matrix and so

$$\begin{aligned} & (P^\pi)^l r^\pi > \tilde{g} - \varepsilon \left((P^\pi)^l - (P^\pi)^{l+1} \right) V_t \\ \therefore \quad & \sum_{l=0}^{T-1} (P^\pi)^l r^\pi > T(\tilde{g} - \varepsilon) + \left(I - (P^\pi)^T \right) V_t \\ \therefore \quad & \underbrace{\frac{1}{T} \sum_{l=0}^{T-1} (P^\pi)^l r^\pi}_{\to g} > \tilde{g} - \varepsilon + \underbrace{\frac{1}{T} \left(I - (P^\pi)^T \right) V_t}_{\to 0} \qquad (1) \\ \therefore \quad & g > \tilde{g} - \varepsilon \end{aligned}$$

For any policy π',

$$\begin{aligned} V_{t+1}(x) &\leqslant Q(x, \pi'(x)) \\ &= r^{\pi'}(x) + P^{\pi'} V_t \end{aligned}$$

Therefore,

$$\begin{aligned} & r^{\pi'}(x) \leqslant V_{t+1}(x) - (P^{\pi'} V_t)(x) \\ \therefore \quad & r^{\pi'} \leqslant (I - P^{\pi'})V_t + V_{t+1} - V_t \\ \therefore \quad & \underbrace{\frac{1}{T} \sum_{l=0}^{T-1} (P^{\pi'})^l r^{\pi'}}_{\to g'} < \tilde{g} + \varepsilon + \underbrace{\frac{1}{T} (I - (P^\pi)^T) V_t}_{\to 0} \qquad (2) \\ \therefore \quad & g' < \tilde{g} + \varepsilon \end{aligned}$$

Per (1) and 2,

$$g > g' - 2\varepsilon$$

□

Logarithmic upper bound on the regret of UCB

Proof. For any $i \in \{1, \dots, K\}$ and $t \geqslant K$, we write $N_t(i)$ the number of plays on machine i after t total plays. For $t > 0$, we also write $\tilde{r}_t(i)$ the average rewards observed after t plays on machine i, and $\hat{r}_t(i)$ the optimist rewards calculated for machine i after t total plays across all machines:

$$\hat{r}_t(i) := \tilde{r}_{N_t(i)}(i) + \sqrt{\frac{2\log t}{N_t(i)}}$$

For the optimal machine[4], these values are written as $\tilde{r}_t^*$, $\hat{r}_t^*$. Note that at any point in time t past the first K plays, the machine that is chosen is the machine i that maximizes the quantity $\hat{r}_t(i)$.

For any event A, we also write the indicator function of A as

$$\{A\} := \begin{cases} 1 & \text{if } A \text{ is true} \\ 0 & \text{otherwise} \end{cases}$$

Finally, we define for any suboptimal machine i

$$l_i := \left\lceil \frac{8\log T}{\Delta_i^2} \right\rceil$$

Then, for $T \geqslant K$, we get the following upper bounds on the number of plays on machine i after T total plays:

$$\begin{aligned}
N_T(i) &= 1 + \sum_{t=K+1}^{T} \{\text{Machine } i \text{ is played at time } t\} \\
&\leqslant l_i + \sum_{t=K+1}^{T} \{\text{Machine } i \text{ is played at time } t \text{ and has been played at least } l_i \text{ times before}\} \\
&\leqslant l_i + \sum_{t=K+1}^{T} \{\hat{r}^*(t-1) \leqslant \hat{r}_i(t-1) \text{ and machine } i \text{ has been played at least } l_i \text{ times before}\} \\
&\leqslant l_i + \sum_{t=K+1}^{T} \left\{ \exists u \in \{1, \dots, t\} \ \exists v \in \{l_i, \dots, t-1\} \ \tilde{r}_u^* + \sqrt{\frac{2\log(t-1)}{u}} \leqslant \tilde{r}_v(i) + \sqrt{\frac{2\log(t-1)}{v}} \right\} \\
&\leqslant l_i + \sum_{t=1}^{\infty} \sum_{u=1}^{t-1} \sum_{v=l_i}^{t-1} \left\{ \underbrace{\tilde{r}_u^* + \sqrt{\frac{2\log t}{u}} \leqslant \tilde{r}_v(i) + \sqrt{\frac{2\log t}{v}}}_{:=A} \right\}
\end{aligned}$$

The event A is only true when at least one of the following events is true:

$$\tilde{r}_u^* \leqslant r^* - \sqrt{\frac{2\log t}{u}} \tag{3}$$

$$\tilde{r}_v(i) \geqslant r(i) + \sqrt{\frac{2\log t}{v}} \tag{4}$$

$$\Delta_i < 2\sqrt{\frac{2\log t}{v}} \tag{5}$$

[4]If two or more machines are tied for best average rewards, consider for example the machine with best average rewards and minimal index.

In other words, either the average rewards of the suboptimal machine have been overestimated, or the average rewards of the optimal machine have been underestimated, or the true averages are close enough. Note that for $v \geqslant l_i$, (5) is always wrong:

$$\begin{aligned} & v \geqslant l_i \\ \implies & v \geqslant \frac{8 \log T}{\Delta_i^2} \geqslant \frac{8 \log t}{\Delta_i^2} \\ \implies & \frac{2 \log t}{v} \leqslant \frac{\Delta_i^2}{4} \\ \implies & 2\sqrt{\frac{2 \log t}{v}} \leqslant \Delta_i \end{aligned}$$

Therefore,

$$N_T(i) \leqslant l_i + \sum_{t=1}^{\infty} \sum_{u=1}^{t-1} \sum_{v=l_i}^{t-1} \left(\left\{ \tilde{r}_u^* \leqslant r^* - \sqrt{\frac{2 \log t}{u}} \right\} + \left\{ \tilde{r}_v(i) \geqslant r(i) + \sqrt{\frac{2 \log t}{v}} \right\} \right)$$

For any $t > 0$, $u \in \{1, \ldots, t-1\}$, $v \in \{l_i, \ldots, t-1\}$ an upper bound on the probability of both events in the above sum can be obtained with the Azuma-Hoeffding inequality:

$$\begin{aligned} \mathbb{P}\left(\tilde{r}_u^* \leqslant r^* - \sqrt{\frac{2 \log t}{u}} \right) &\leqslant \frac{1}{t^4} \\ \mathbb{P}\left(\tilde{r}_v(i) \geqslant r(i) + \sqrt{\frac{2 \log t}{v}} \right) &\leqslant \frac{1}{t^4} \end{aligned}$$

Thus,

$$\begin{aligned} \mathbb{E}\left(N_T(i)\right) &\leqslant l_i + \sum_{t=1}^{\infty} \sum_{u=1}^{t-1} \sum_{v=l_i}^{t-1} \left(\frac{2}{t^4} \right) \\ &\leqslant l_i + 2 \sum_{t=1}^{\infty} \frac{1}{t^2} \\ &= \left\lceil \frac{8 \log T}{\Delta_i^2} \right\rceil + \frac{\pi^2}{3} \end{aligned}$$

This holds true for any machine i such that $r(i) \neq r^*$ and therefore

$$
\begin{aligned}
\mathrm{Reg}(T) &= \mathbb{E}\left(\sum_{i=1}^{K} N_T(i)\Delta_i\right) \\
&= \sum_{i:r_i\neq r^*} \mathbb{E}\left(N_T(i)\right)\Delta_i \\
&\leqslant \sum_{i:r_i\neq r^*} \left(\left\lceil \frac{8\log T}{\Delta_i^2}\right\rceil + \frac{\pi^2}{3}\right)\Delta_i \\
&\leqslant \sum_{i:r_i\neq r^*} \left(\frac{8\log T}{\Delta_i} + \left(1+\frac{\pi^2}{3}\right)\Delta_i\right) \\
&= 8\sum_{i:r_i\neq r^*} \frac{\log T}{\Delta_i} + \left(1+\frac{\pi^2}{3}\right)\left(\sum_{i=1}^{K}\Delta_i\right)
\end{aligned}
$$

□

B Institutional context

This internship was completed during a six week period among the POLARIS team at the Laboratoire d'Informatique de Grenoble (LIG), a computer science laboratory under the joint supervision of the Université Grenoble Alpes (UGA), the Institut polytechnique de Grenoble (Grenoble INP), and the CNRS, and a partner of the Inria.

The POLARIS team aims to *"contribute to the understanding (from the observation, modeling and analysis to the actual optimization through adapted algorithms) of the performance of very large scale distributed systems by applying original ideas from various research fields and application domains"*; Domains of study include measurement and experimentation, statistical trace analysis and visualization, discrete-event and perfect simulation, asymptotic modeling, stochastic optimization and game theory.

This first internship was a unique experience and introduction to the world of research in computer science, allowing me to get an idea of the career of research in computer science and of the social life in research laboratories, and giving me the opportunity to interact with interns and researchers of the other teams working at the LIG, on themes such as network programming and computer-assisted proof. Furthermore, I was able to present my work to the Polaris-TT conference near the conclusion of the internship, also giving me the opportunity to learn about the work of a fellow intern on the team who studied a similar subject.