On the road from Model-Based Dynamic Programming to Model-Free Reinforcement Learning

A sample efficient approach

Pol Mur i Uribe

Master of Science Thesis

TU Delft Delft University of Technology

Delft Center for Systems and Control

On the road from Model-Based Dynamic Programming to Model-Free Reinforcement Learning

A sample efficient approach

Master of Science Thesis

Pol Mur i Uribe

March 15, 2023

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Abstract

This thesis introduces a new method, called Mixed Iteration, for controlling Markov Decision Processes when partial information is known about the dynamics of the Markov Decision Process. The algorithm uses sampling to calculate the expectation of partially known dynamics in stochastic environments. Its goal is to lower the number of iterations and computational steps required for convergence compared to traditional model-free algorithms. By lowering the number of samples required to achieve convergence Markov Decision Processes can be controlled and trained more efficiently. Additionally, the thesis discusses how this algorithm can enhance the sample efficiency and convergence rate of Reinforcement Learning algorithms like Q-Learning. The effectiveness of the proposed method will be evaluated in standard Reinforcement Learning problems and compared with the performance of Q-learning. The results show that under certain conditions that will be discussed in the thesis, the new proposed algorithm outperforms classical algorithms in terms of sample efficiency. The study will provide insight into the field of previous partial information in Reinforcement Learning alternatives, as well as the challenges that researchers in this field continue to face.

Keywords: Markov Decision Process (MDP), Reinforcement Learning, partial information, sample efficiency.

Table of Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank my supervisor Peyman Mohajerin Esfahani for his assistance during the writing of this thesis and the support during these months. I would also like to thank Amin Sharifi Kolarijani and Pedro L. Ferreira for all the meetings and knowledge you shared with me. Lastly, to all my friends and study partners that shared with me uncountable hours of studying and finally, to you Lara.

Delft, University of Technology
March 15, 2023

Pol Mur i Uribe

Chapter 1

Introduction

Stochastic processes were mathematically defined in the early 1930s by Aleksandr Khinchin [4] as a probability model describing a collection of time-ordered random variables that represent the possible sample paths. Stochastic Processes can be grouped into different categories being one of them Markov Processes introduced by Andrey Markov [5]. A Markov Process is described as a stochastic model that describes a sequence of events in which the probabilities of future events are determined by the previous state [6]. Markov Decision Processes (MDP) are an extension of Markov Processes where the transition between the states is influenced by the action chosen by a decision maker. Markov Decision Processes are known since 1950s after the paper published by Richard Bellman in 1954 [7]. Markov Decision Processes are a powerful tool used in the optimization of the actions determined by the decision maker.

MDPs future states are determined by the combination of the action chosen by the decision maker with a stochastic transition probability distribution. Depending on the amount of prior knowledge regarding the transition probability distribution, model-free or model-based [8] algorithms must be applied to optimize the choice of the sequence of future actions. Model-based algorithms [9] are any MDP approach that uses a model previously known or learned. On the other hand, model-free algorithms are used without the explicit model of the environment. Model-free methods [10] abstain from learning the model and directly determine a value to either a state or a state-action pair.

Richard Sutton formalized Reinforcement Learning (RL) in 1984 during his PhD thesis [11] as an area of artificial intelligence where an agent takes actions in an environment, usually described by an MDP, in order to optimize an objective function. Reinforcement Learning is used in several research fields including robotics, energy management, manufacturing, and economics [12]. Another research area where RL is widely used are games, where algorithms have been developed since the 1990s, when TD-Gammon was introduced for solving the Backgammon game [13]. Another successful implementations of similar algorithms are AlphaZero [14], for Chess and Go, and AlphaStar [15] for Starcraft II, both algorithms developed by DeepMind.

The duality between model-based and model-free algorithms will allow the combination of them in certain scenarios aiming for an improvement of the metrics that determine their success. The aim of this thesis will be to find an efficient combination of model-free and model-based methods to reduce the sample complexity and improve the convergence of the algorithms under different environments. By reducing the sample complexity, the number of iterations required can be decreased significantly in complex environments. In some applications where every iteration is highly costly, diminishing the number of iterations required to obtain an optimal solution will have a substantial impact.

In Chapter 2 the concept of Markov Decision Processes (MDP) will be discussed as well as the problem that needs to be solved when dealing with MDPs. In the following Chapter 3, the difference between model-free and model-based algorithms will be studied when trying to solve an MDP as well as the gap between the two algorithms and the problem statement will be posed. In Chapter 4, the alternatives that are currently present in the literature will be discussed for solving the problem of the gap in performance between model-free and model-based algorithms. The solution proposed in this thesis is presented in Chapter 5, where the newly proposed algorithm is described and the principles behind it as well as convergence guarantees. The next step in the thesis will be to design the experiments where the algorithm will be tested, simulate the algorithm and obtain results, this will be done in Chapter 6. Finally, in Chapter 7 a further analysis of the results will be done followed by the limitations found for the algorithm proposed and directions for future research will be appointed.

Chapter 2

Markov Decision Processes

MDPs are a discrete- or continuous-time stochastic control process where the Markovian property is kept along the states. MDPs provide a framework for decision-making where the transition between the states is partially random and influenced by a decision-maker or controller. In this chapter, the definitions of Markov Decision Processes and the elements that compose the Markov Process framework will be explained. The scope of the thesis will be restricted to discrete-time stochastic processes and all the definitions provided during the next sections will assume a finite state-action space.

2-1 Markov Processes

Markov Processes are stochastic processes that in a discrete-time setting the transition probabilities between the states are defined as follows.

Definition 2.1. *A stochastic process $X = \{X_t : t \geq 0\}$ is a Markov Process if*

$$\mathbb{P}\{X_{t+1}|X_0, X_1, ..., X_t\} = \mathbb{P}\{X_{t+1}|X_t\}, \tag{2-1}$$

Moreover, for a time-homogeneous Markov Chain,

$$\mathbb{P}\{X_{t+1} = j|X_t = i\} = P_{i,j} \quad \forall i, j \in \mathbb{X}, \forall t \geq 0. \tag{2-2}$$

The first equation in the definition is the Markov Property that states that at any step t, the probability of $X_{t+1}$is only dependent on the current state X_t and independent of the previous states. In the case of time-homogeneous transition probabilities then the second condition states that the probabilities are homogeneous for all steps t and only depend on the current state. An extension of Markov Processes are Markov Decision Processes, where at every step t a decision-maker chooses an action in order to optimize an objective function according to the dynamics described by the transition probability matrix $\mathbb{P}$.

2-2 Markov Decision Processes

Markov Decision Processes are defined as a discrete-time stochastic control process where at every time step an action needs to be chosen and the outcome is stochastic. MDP theory is applied to control of Markov Processes.

A Markov Decision Process is fully described with the elements in the tuple $\mathcal{M} = (\mathbb{X}, \mathbb{U}, \mathbb{P}, c_{x,u}, \gamma)$:

- The set $\mathbb{X}$ is the state space of finite size $|\mathbb{X}| = n$.
- The set $\mathbb{U}$ describes the action space of finite size $|\mathbb{U}| = m$.
- The function $c : \mathbb{X} \times \mathbb{U} \to \mathbb{R}$ is the cost function associated with a state-action pair.
- The transition probability kernel $\mathbb{P}$ defines the probability of landing in the next state given the current state and the action chosen by the decision-maker, that is,

$$x_{t+1} \sim \mathbb{P}\left(\cdot \mid x_t,\, u_t\right). \tag{2-3}$$

- The discount factor $\gamma \in (0, 1)$ will determine how much the distant future cost will be considered compared to the immediate costs.

2-3 Solving a Markov Decision Process

The objective of describing an environment as a Markov Decision Process is to optimize the sequence of future actions according to a policy π. The objective function that will be minimized will be the expected sum of discounted costs during an infinite horizon of time. The optimal sequence of actions will be obtained after finding the policy $\pi : \mathbb{X} \to \mathbb{U}$ that minimizes the sum of discounted costs over an infinite horizon of time.

The optimal cost will be the solution obtained after applying the optimal policy π^*, the minimizer of the optimization problem below

$$J^*(x) = \min_{\pi:\mathbb{X}\to\mathbb{U}} \mathbb{E}_{\mathbb{P}}\left[\sum_{k=0}^{\infty} \gamma^k c\left(X_k, \pi\left(X_k\right)\right) \mid X_0 = x\right] \quad \forall x \in \mathbb{X}. \tag{2-4}$$

The standard solution for the problem posed is the solution of the next fixed-point equation

$$J^* = \mathcal{T} J^*. \tag{2-5}$$

To achieve the desired solution, consecutive applications of the Bellman operator will be performed until the fixed-point solution is reached. The Bellman Operator $\mathcal{T}$,

$$\mathcal{T} J(x) := \min_{u\in\mathbb{U}} \quad \left\{c(x,u) + \gamma \, \left[\mathbb{E}_{x^+\sim\mathbb{P}(\cdot|x,u)}[J(x^+)\,]\right]\right\} \quad \forall x \in \mathbb{X}, \tag{2-6}$$

by applying the Bellman Operator recursively the cost $J(x)$ will be computed for all x updating the values of the entire vector J,

$$J_{k+1} = \mathcal{T} J_k\,. \tag{2-7}$$

The Bellman Operator is known to be a contractor [16]. This means that the following property applies to the iterations with $\gamma < 1$, where the infinity-norm is defined as $\|\mathbf{x}\|_\infty := \max_i |x_i|$,

$$\|\mathcal{T}J - \mathcal{T}J'\|_\infty \leq \gamma \|J - J'\|_\infty. \tag{2-8}$$

The previously mentioned contraction property ensures that for any starting point, the iterations will converge to a unique fixed point $J^*(x)$. One of the main challenges of the equation proposed above is how to solve the expectation with respect to $\mathbb{P}$, as the probability distribution is often unknown or partially known. Depending on the amount of prior knowledge on $\mathbb{P}$ we can divide the approaches between model-based, when $\mathbb{P}$ is fully known, or model-free in the rest of cases. In both scenarios the existing literature provides algorithms for model-free and model-based cases. However, there are several differences in the performance as in the model-based case all the information is available making the process of solving the expectation simpler than in the model-free scenario.

Chapter 3

Model-Based and Model-Free Algorithms

During this chapter, the nature of model-based and model-free algorithms will be discussed as well as their performances. When analyzing their performances, we will highlight the gap in the existing literature and the reasons why the proposed algorithm could help close this gap. The algorithms presented in this chapter are the simplest first-order algorithms in both model-based and model-free algorithms, namely, Value Iteration [17] for model-based and Q-Learning [18] for model-free. The chosen algorithms despite being the simplest show the differences between the two approaches and the same comparison could be made with second-order algorithms of both families or their respective accelerated versions. However, in order to make a fair comparison we should always compare algorithms of the same nature between model-free and model-based algorithms.

3-1 Model-Based Value Iteration

Value Iteration was introduced by Bellman in 1957 [19] and simply involves consecutive applications of the Bellman operator as in $J_{k+1}(x) = \mathcal{T} J_k(x)$. The iteration is typically performed until two steps are smaller than a certain threshold ϵ, i.e.,

$$\max_{x \in \mathbb{X}} |J_k(x) - J_{k+1}(x)| < \epsilon. \tag{3-1}$$

Value Iteration can be used in the case when the transition probability tensor, $\mathbb{P}$, is fully known:

$$J_{k+1}(x) := \min_{u \in \mathbb{U}} \ \Big\{ c(x,u) + \gamma \, \mathbb{E}_{x^+ \sim \mathbb{P}(\cdot|x,u)}[J_k(x^+)] \Big\} \quad \forall x \in \mathbb{X}. \tag{3-2}$$

In particular, in the case of finite state space, the expectation operation reduces to

$$\mathbb{E}_{x^+ \sim \mathbb{P}(\cdot|x,u)}[J_k(x^+)\,] = \sum_{x^+ \in \mathbb{X}} \mathbb{P}(x^+ \mid x, u) J_k(x^+). \tag{3-3}$$

The algorithm is guaranteed to converge to the unique fixed-point J^* of the Bellman operator $\mathcal{T}$, that is, $J_k \to J^*$ as $k \to \infty$ if $0 < \gamma < 1$. The preceding result follows from Banach Fixed Point theorem [20] and the fact that the Bellman operator is a contraction with coefficient γ, that is, for any two cost functions J_1 and J_2,

$$||\mathcal{T}(J_1) - \mathcal{T}(J_2)||_\infty \leq \gamma ||J_1 - J_2||_\infty. \tag{3-4}$$

The optimal policy is then the greedy policy with respect to J^* given by,

$$\pi^*(x) = \arg\min_{u \in \mathbb{U}} \quad \left\{ c(x,u) + \gamma \left[\sum_{x^+ \in \mathbb{X}} \mathbb{P}(x^+ \mid x, u) J^*(x^+) \right] \right\} \quad \forall x \in \mathbb{X}. \tag{3-5}$$

Finally, the cost function J^π will be defined as the cost function obtained if the policy π is followed,

$$J^\pi(x) := \mathbb{E}_{\mathbb{P}} \left[\sum_{k=0}^{\infty} \gamma^k c\left(X_k, \pi\left(X_k\right)\right) \mid X_0 = x \right] \quad \forall x \in \mathbb{X}. \tag{3-6}$$

The same notation can be used for any other policies that could be obtained, so for the optimal policy π^*, the cost J^{π^*} will be the same result of Equation 3-6. It can be also defined after Equations 3-6 and 3-5 that $J^* = J^{\pi^*}$.

3-1-1 The Value Iteration Algorithm

The Value Iteration algorithm is presented next and it is guaranteed to converge as was discussed in the previous section. The inputs required for the algorithm will be explained as follows. The first variable ϵ will be the convergence threshold that will determine when the error between two consecutive iterations is close enough, $\mathbb{X}$ and $\mathbb{U}$ will be the possible set of states and actions that comprise the state-action space, $c(x,u)$ determines the stage cost of choosing a particular action being at a particular state, $\mathbb{P}$ is the transition probability tensor that defines the dynamics of the MDP, and finally γ is the discount factor between 0 and 1.

Algorithm 1 Value Iteration

Input: ϵ, $\mathbb{X}$, $\mathbb{U}$, $c(x,u)$, $\mathbb{P}$, γ
Output: ϵ-optimal policy
  for $x \in \mathbb{X}$ **do**
    $J_0(x) \leftarrow \min_u c(x,u)$
  end for
  while $\max_{x \in \mathbb{X}} |J_k(x) - J_{k+1}(x)| > \frac{\epsilon}{1-\gamma}$ **do**
    for $x \in \mathbb{X}$ **do**
      $J_{k+1}(x) := \min_{u \in \mathbb{U}} \quad \{c(x,u) + \gamma \ [\sum_{x^+ \in \mathbb{X}} \mathbb{P}(x^+ \mid x, u) J_k(x^+)]\}$
    end for
  end while
  $\pi(x) = \arg\min_{u \in \mathbb{U}} \quad \{c(x,u) + \gamma \ [\sum_{x^+ \in \mathbb{X}} \mathbb{P}(x^+ \mid x, u) J(x^+)]\} \quad \forall x \in \mathbb{X}$

3-1-2 Convergence Guarantee of Value Iteration

In Appendix A the Banach fixed-point theorem is discussed, it ensures that the Bellman Equation has a unique solution and is guaranteed to converge due to the contraction property the Bellman Operator has. We will discuss the effect of γ and derive a bound for the number of iterations needed to converge within ϵ. An ϵ-optimal value function will be any value function J such that $\|J - J^*\| \leq \epsilon$. When obtained, the greedy policy with respect to J can be obtained so an ϵ-optimal policy can be derived. It has been discussed [21] that given the iterative nature of the algorithm the closer to 1 the value of γ is, the slower the convergence will be. This is because the future values will be less discounted and will be taken more into account, thus needing a long-term horizon view to derive the optimal policy.

Then it can be derived that for any $J(x)$ a finite number of iterations is needed to reduce the error between the current cost and $J^*(x)$ so an ϵ-optimal cost function is found. The number of iterations can be derived as follows. Assuming that the costs are bounded

$$|c(x,u)| \leq c_{max} \quad \forall x \in \mathbb{X}, \forall u \in \mathbb{U}. \tag{3-7}$$

Then applying the property of geometric series,

$$\begin{aligned} -\frac{c_{max}}{1-\gamma} \;&\leq\; J(x) \;\leq\; \frac{c_{max}}{1-\gamma} \\ ||J(x) - J^*(x)||_\infty \;&\leq\; \frac{2\,c_{max}}{1-\gamma}. \end{aligned} \tag{3-8}$$

Then the upper-bound on the number of iterations k needed to guarantee convergence within ϵ will be the value that satisfies,

$$\begin{aligned} \gamma^k \frac{2\,c_{max}}{1-\gamma} \;&\leq \epsilon \\ k \;&\leq \left\lceil \frac{\log\left[\frac{2\,c_{max}}{\epsilon(1-\gamma)}\right]}{\log\left(\frac{1}{\gamma}\right)} \right\rceil. \end{aligned} \tag{3-9}$$

According to the expression derived for the number of necessary iterations to converge within ϵ it can be observed that the number of iterations increase in the order of $\log(\frac{1}{\epsilon})$. Also, the closer γ is to 1 then the number of iterations will also increase rapidly.

In order to derive the computational complexity, apart from determining the number of iterations needed to converge it needs to be derived the per-iteration complexity. In the case of Value Iteration, the complexity of every iteration depends on the space of the state-action space. To complete one iteration, for each state, it is needed to minimize over all possible actions while computing the objective requires computing the expectation with respect to all possible future states. Then, if n is the size of the state space and m is the size of the action space, the iteration complexity of Value Iteration will be $\mathcal{O}(n^2 m)$.

3-2 Model-Free Q-Learning

An alternative approach to minimizing the expected value of the infinite sum of discounted rewards when $\mathbb{P}$ was unknown or partially known was introduced in 1989 by Chris Watkins

[22]. The algorithm proposed is named Q-Learning and solves the problem of computing the expectation via sampling the system. In the Q-learning algorithm, a cost function will be associated with every state-action pair, $Q(x,u)$. The goal of the algorithm will be the same as in the model-based case, finding an optimal policy for every possible state. The algorithm is guaranteed to converge into the optimal policy if all state-action pairs are visited infinitely often [23]. So the goal of Q-Learning will be to achieve Q^* such that,

$$Q^*(x,u) = c(x,u) + \gamma\, \mathbb{E}_{x^+\sim\mathbb{P}(\cdot|x,u)} \left[J^*(x^+,u^+) \right]. \tag{3-10}$$

Q-Learning is an iterative algorithm that will be performed until the difference between iterations is below a certain value ϵ. Ideally, with an infinite amount of time and visiting every state-action pair it is guaranteed to converge, however, once the ϵ-optimal value function has been found then the greedy-policy $\pi(x)$ respect the optimal Q-function will be obtained,

$$\pi(x) = \arg\min_{u\in\mathbb{U}} Q(x,u). \quad \forall x \in \mathbb{X} \tag{3-11}$$

If we combine Equation 3-10 with the Bellman Equation 2-5 then one can also derive the following relation,

$$J(x) = \min_{u\in\mathbb{U}} Q(x,u) \quad \forall x \in \mathbb{X}, u \in \mathbb{U}. \tag{3-12}$$

The expectation described in Equation 3-10 with respect to $\mathbb{P}$ cannot be computed directly in the model-free case as there will be a lack of knowledge about $\mathbb{P}$. To compute the value of the expectation another approach will be used. By sampling the system and receiving the information of the next state $x^+ \sim \mathbb{P}(\cdot|x,u)$ we can solve the expectation with respect to $\mathbb{P}$. The approximation of the mentioned expectation follows a similar principle to the Stochastic Gradient Descend algorithm. To prove the convergence of the algorithm and the relation between the Bellman Operator and Q-Learning, we will introduce the estimator of the Bellman operator. The estimator of the Bellman Operator $\hat{\mathcal{T}}$ will be defined as an estimator of $\mathcal{T}$,

$$\hat{\mathcal{T}}Q(x,u,x^+) := c(x,u) + \gamma \min_{u^+\in\mathbb{U}} Q(x^+,u^+) \tag{3-13}$$

The estimator $\hat{\mathcal{T}}$ will converge to $\mathcal{T}$ with enough sampling of the system and visiting all the possible states infinitely often. The Bellman Operator as described in Equation 3-2 is defined as,

$$\mathcal{T}Q(x,u) := c(x,u) + \gamma\, \mathbb{E}_{\mathbb{P}} \left[\min_{u^+\in\mathbb{U}} Q(x^+,u^+) \right] \quad \forall x \in \mathbb{X}, \forall u \in \mathbb{U} \tag{3-14}$$

as a consequence of the definition of $\hat{\mathcal{T}}$ through enough sampling and visiting all the state-action pairs infinitely often, for any fixed $Q(x,u)$ the estimator will converge to the Bellman operator

$$\mathbb{E}_{\mathbb{P}} \left[\hat{\mathcal{T}}Q(x,u,x^+) \right] = \mathcal{T}Q(x,u) \quad \forall x \in \mathbb{X}, \forall u \in \mathbb{U}. \tag{3-15}$$

As it was derived in the previous section when the Bellman Operator was discussed the approximate Bellman operator also suffices the Banach Fix Point theorem as the contraction property also applies [24],

$$||\hat{\mathcal{T}}Q(x,u) - \hat{\mathcal{T}}Q^*(x,u)||_\infty \leq \gamma ||Q(x,u) - Q^*(x,u)||_\infty \quad \forall x \in \mathbb{X}, \forall u \in \mathbb{U}. \tag{3-16}$$

The learning rate typically used in Q-Learning, α is a polynomial learning rate that decreases over the iterations $\frac{1}{k^\omega}$ where $\omega \in (\frac{1}{2}, 1)$ [25]. The values of $Q(x,u)$ will be obtained with the next update rule,

$$Q_{k+1}(x,u) = Q_k(x,u) + \alpha\left(\hat{\mathcal{T}}Q_k(x,u,x^+) - Q_k(x,u)\right) \quad \forall x \in \mathbb{X}, u \in \mathbb{U}. \tag{3-17}$$

3-2-1 The Q-Learning Algorithm

The Q-Learning algorithm that will be used in future sections and used as a reference is the algorithm shown next. It can be observed in the algorithm the needed inputs for it to work and after iterating the optimal policy will be obtained as the process is guaranteed to converge. The inputs required for the algorithm will be explained as follows. The first variable ϵ will be the convergence threshold that will determine when the error between two consecutive iterations is close enough, $\mathbb{X}$ and $\mathbb{U}$ will be the possible set of states and actions that comprise the state-action space, $c(x,u)$ determines the stage cost of choosing a particular action being at a particular state, $\mathbb{P}$ is the transition probability tensor that defines the dynamics of the MDP, and finally γ is the discount factor between 0 and 1. The convergence is guaranteed however it might take a significant number of states, for this reason a maximum number of iterations k_{max} will be also added to the algorithm.

Algorithm 2 Q-Learning

Input: ϵ, $\mathbb{X}$, $\mathbb{U}$, $c(x,u)$, γ, k_{max}
Output: π, the sub-optimal policy
  $Q_0(x,u) \leftarrow \mathbf{0}_{n\times m}$
  while $\max_{(x,u)} |Q_k(x,u) - Q_{k+1}(x,u)| > \epsilon$ and $k < k_{max}$ **do**
    for $x \in \mathbb{X}, u \in \mathbb{U}$ **do**
      $x^+ \sim \mathbb{P}(\cdot \mid x,u)$
      $Q_{k+1}(x,u) = Q_k(x,u) + \alpha\left(\hat{\mathcal{T}}Q_k(x,u,x^+) - Q_k(x,u)\right)$
    end for
  end while
  $\pi(x) = \arg\min_{u\in\mathbb{U}} \quad Q(x,u)$

3-2-2 Convergence Guarantee of Q-Learning

The discussion of the bounds for convergence of Q-Learning has been addressed for years by the Reinforcement Learning community under different conditions. In this section, the iteration complexity will be discussed for the cases of linear and polynomial step sizes. The first results on the convergence of the algorithm were presented in the late 90s [26] [27] and showed that the convergence rate of Q-Learning is strongly linked to the learning rate in the order of $\frac{1}{1-\gamma}$. The bounds for the Q-Learning algorithm for the non-rescaled linear step size $\alpha_k = k^{-1}$ and the polynomial step-size $\alpha_k = k^{-\omega}$ for $\omega \in (0,1)$ were derived by Even-Dar and Mansour [25] and will be the bounds considered as the algorithm will be implemented with polynomial step sizes. For all the sequence of step sizes considered they need to meet two criteria to ensure convergence,

$$\sum_k \alpha_k = \infty \quad \sum_k \alpha_k^2 \leq \infty. \tag{3-18}$$

The type of MDPs considered for this analysis will be cost-bounded, that is

$$|c(x,u)| \leq c_{max}, \quad \forall x \in \mathbb{X}, \forall u \in \mathbb{U} \tag{3-19}$$

and the iteration complexity will be the number of iterations required to reduce the expectation of the infinity norm $\mathbb{E}||Q_k - Q^*||_\infty$ below ϵ. Notice that with Q-Learning the per-iteration complexity is of the order of $\mathcal{O}(m^2 n)$, where m and n are the sizes of the action and state spaces respectively. The total computational complexity will be the one derived in the next subsections times the per-iteration complexity.

Linear Step Sizes

The first results regarding the iteration complexity of unrescaled step sizes were quite pessimistic exponentially growing in the order of $\frac{1}{1-\gamma}$ [24]. However, if the rescaled version of the step size is used $\alpha_k = \frac{1}{1+(1-\gamma)k}$ then the iteration complexity grows polynomially instead of exponentially [25]. To derive the expression for the bounds in the number of iterations, there are two variables that need to be explained. The first one is the span seminorm of Q^* described as,

$$||Q^*||_{\text{span}} = \max_{(x,u)} Q^*(x,u) - \min_{(x,u)} Q^*(x,u). \tag{3-20}$$

And the second one will be the maximal standard deviation. As the iteration process will be i.i.d. and zero mean the variance can be computed easily and the value of the maximal standard deviation will be defined as

$$\|\sigma(Q^*)\|_\infty = \sqrt{\max_{(x,u)} \sigma^2(Q^*)(x,u)}. \tag{3-21}$$

After having explained these two variables Wainwright [24] derived the iteration complexity for the rescaled linear step size,

$$k \precsim \left(\frac{\|Q_0 - Q^*\|_\infty}{1-\gamma} + \frac{\|Q^*\|_{\text{span}}}{(1-\gamma)^2} \right) \left(\frac{1}{\epsilon} \right) + \left(\frac{\|\sigma(Q^*)\|_\infty^2}{(1-\gamma)^3 \epsilon^2} \right) \tag{3-22}$$

iterations, convergence can be guaranteed within ϵ. The notation $\precsim$ means that the derived equation is a simplification that holds after dropping logarithmic factors. The main concern that will be considered when designing the parameters that will determine the success of the convergence will be the choice of the parameter ϵ. If the rest of the parameters are considered constant across all design scenarios then we can drop them and the resulting iteration complexity will be dependent on ϵ in the order of,

$$k \precsim \tilde{O}\left(\frac{1}{\epsilon^2} \right). \tag{3-23}$$

Polynomial Step Sizes

In the case of Q-Learning with polynomial step sizes Even-Dar and Mansour [25] in their Theorem 2 they proved that it suffices to take at most

$$k \precsim \left(\frac{c_{\max}^2}{(1-\gamma)^4 \epsilon^2} \right)^{\frac{1}{\omega}} + \left\{ \frac{1}{1-\gamma} \log \left(\frac{c_{\max}}{(1-\gamma)\epsilon} \right) \right\}^{\frac{1}{1-\omega}} \tag{3-24}$$

iterations to bring the infinity norm of the error of any MDP with bounded rewards below a certain threshold ϵ. As we did in the case of rescaled linear step sizes for the polynomial step sizes the dependency on ϵ will be,

$$k \precsim \tilde{O}\left(\frac{1}{\epsilon^{\frac{2}{\omega}}}\right). \tag{3-25}$$

3-3 The gap between Value Iteration and Q-Learning

During this chapter, the principles and derivations of Value Iteration and Q-Learning were explained. When both algorithms are implemented there are obvious differences in their performances that should be highlighted. The algorithms will be implemented in a simple MDP called the `RiverSwim` MDP [28] with six states and two actions. The probabilities of the transitions as well as the costs associated with the states are shown in Figure 3-1.

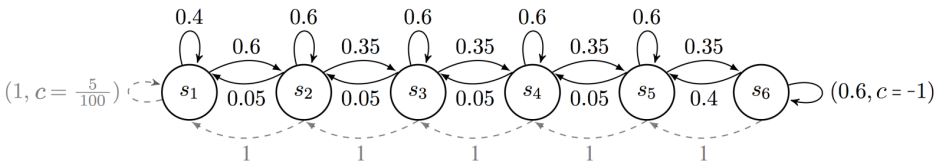


Figure 3-1: `RiverSwim` MDP – solid and dotted arrows denote the transitions under actions 'right' and 'left', respectively [1]

When trying to solve the MDP and obtain the optimal policy as well as the optimal values of $J^*(x)$ and $Q^*(x, u)$, it can be observed that Value Iteration converges faster than Q-Learning as shown in Figure 3-2.

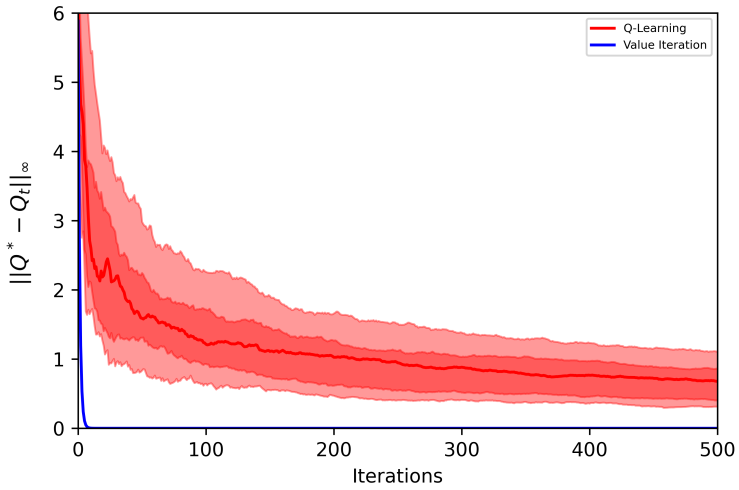


Figure 3-2: In thick red the average of the evolution of the error between the value of Q_k and Q^* over 50 runs. In shadowed red 10%-90% and 25%-75% confidence intervals, using Value Iteration and Q-Learning for the `RiverSwim` MDP case along 500 iterations

As Q-Learning has some stochasticity associated due to the random sampling of the next state, a series of 50 runs were performed and the results shown in the plot are the mean and the 10%-90% as well as the 25%-75% percentiles. When we compare the performance of both algorithms, Value Iteration converges after very few iterations while Q-Learning takes a significant number of iterations to converge to the optimal cost function. The fact that Value Iteration converges faster than Q-Learning is due to the order of the polynomial convergence guarantee as it is higher in Q-Learning than in Value Iteration. In the case of Value Iteration, the expectation of future costs is computed analytically while in Q-Learning we have to rely on solving the expectation through sampling. If we want to find the exact value of the expectation infinite sampling and visits to every state-action pairs are required and sometimes this is hard to achieve. After discussing the differences between the two algorithms a question arises: How could this gap be filled in the cases where partial information is known? Is it possible to solve the expectation including partial knowledge about the transitions to make the process converge faster? In some reinforcement learning applications, computational complexity is crucial as every iteration can be associated with a cost per iteration so reducing the number of iterations needed to provide an accurate control policy will reduce the cost of implementing the algorithm.

The algorithm to be designed must:

- Ensure convergence under the same conditions required in Q-Learning and Value Iteration solving the fixed-point equation discussed.
- Reduce the number of computations needed to obtain reliable solutions making the algorithm more sample-efficient than Q-Learning.
- Be able to converge under any percentage of previously known probability transitions.

Once the problem has been formulated, the coming chapters of the thesis will focus on exploring different alternatives that are found in Literature and analysing them. It will also be further discussed the exact goal of the algorithm proposed. Lastly, the algorithm will be tested in benchmark Reinforcement Learning problems and the results obtained will be compared to the performance of Q-Learning. The metrics used to compare the success of the algorithm will be also discussed.

3-4 Accelerated Versions of Q-Learning

The goal of the thesis and the algorithm proposed will not be to accelerate the speed of convergence of Q-Learning. The goal will be to propose a framework to combine model-based and model-free algorithms so the methodology can be implemented to faster versions of Q-Learning in future work. If we would like to reduce the number of iterations needed to converge to the optimal Q^*, algorithms like Double Q-Learning or Speedy Q-Learning could be used. In this subsection, the algorithms mentioned will be briefly discussed so that if needed in future work we could use the algorithm that will be proposed to combine the accelerated versions of Q-Learning with Value Iteration as well.

Double Q-Learning

Double Q-Learning aims to implement a double estimation method resulting in fewer iterations to converge in situations where the Q-Learning algorithm overestimates the maximum expected cost due to the minimization computed in Q-Learning. The idea of the algorithm will be to have two different Q functions and one of them will be updated at every iteration using crossed information between the two Q functions. This method was introduced in [29] when the Reinforcement Learning community was trying to speed up the convergence of model-free RL algorithms. This algorithm has been extensively used in Deep Learning applications where the results have proven the Double Q-Learning to be faster than the original Q-Learning. However, this algorithm has the problem that the values needed to store are twice as large as in Q-Learning. Thus in large state-action spaces, it can be difficult to store and update the two cost functions to ensure that the algorithm converges to Q^*.

Speedy Q-Learning

Speedy Q-Learning is based on the Q-Learning updates but in the Speedy version, the algorithm keeps track of the two last updates of the function $Q(x,u)$. The paper proposed states that in the cases where γ is closer to 1 Speedy Q-Learning outperforms standard Q-Learning as the convergence bounds derived do not depend as much as in the standard Q-Learning on the $\frac{1}{1-\gamma}$ factor. The idea of the newly proposed updates is that despite requiring $2nm$ the amount of memory for the updates the algorithm will be more sample-efficient and will converge to the optimal state-action cost function. To describe the Speedy Q-Learning algorithm the approximate Bellman Operator introduced in previous sections $\hat{\mathcal{T}}Q(x,u,x^+) = c(x,u) + \gamma \min_{u^+ \in \mathbb{U}} Q(x^+,u^+)$ will be used. Then, the updates of the Q function will be done as follows,

$$Q_{k+1}(x,u) = Q_k(x,u) + \alpha_k(\hat{\mathcal{T}}Q_{k-1}(x,u,x^+) - Q_k(x,u)) + \alpha_k(\hat{\mathcal{T}}Q_k(x,u,x^+) - \hat{\mathcal{T}}Q_{k-1}(x,u,x^+)). \tag{3-26}$$

The algorithm performs well in generative models if exploration is not considered. Also, it is guaranteed to converge and the convergence bounds in [30] show that the algorithm performs better than standard Q-Learning.

Chapter 4

Alternatives in the Literature

In Chapter 4, different alternatives to solve the problem of uncertainty in the transition probability matrix will be discussed. The approaches proposed in Literature are based on Robust Optimization, Transfer Learning and System Identification.

4-1 Robust Reinforcement Learning

Robust Reinforcement Learning provides an alternative to estimations of the transition probability matrix. As the transition probability matrix is often not fully available or estimated through biased experiments under certain conditions, the learned policy might not be suitable for all possible environments as these can vary from the training environments [31].

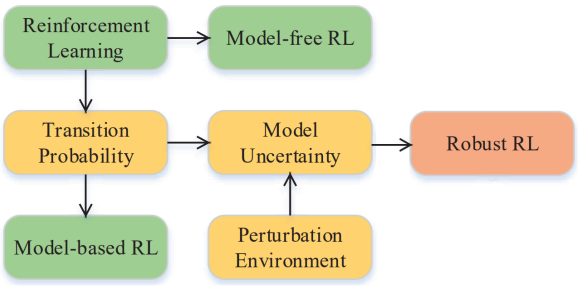


Figure 4-1: Derivation of Robust RL solving the uncertainty when the transition probability matrix is not known [2]

The formulation of the problem will aim to reduce the sensitivity to uncertainty of the optimal policy. The robust RL problem will learn a robust optimal policy with unknown transition dynamics [32]. To do so, we have to define a set of transition probability matrices, $\mathcal{P}$. Then, the cost function of Robust Reinforcement Learning will be

$$J^{\pi}(x) = \sup_{\mathbb{P}\in\mathcal{P}} \mathbb{E}_{\mathbb{P}}\left[\sum_{k=0}^{\infty} \gamma^k c\left(X_k, \pi\left(X_k\right)\right) \mid X_0 = x\right]. \quad \forall x \in \mathbb{X} \tag{4-1}$$

The controller will choose a policy that will minimize the cost of the worst-case scenario with the robust Bellman equation [2],

$$J_{k+1}\left(x\right) = \inf_{u\in\pi} \sup_{\mathbb{P}\in\mathcal{P}} \mathbb{E}_{\mathbb{P}}\left[c\left(x, \pi(x)\right) + \gamma J_k\left(x^{+}\right)\right] \tag{4-2}$$

The uncertain sets of transition probabilities $\mathcal{P}$ can be described using likelihood models, entropy models or KL-divergence models among others [33],[34]. The robust approach is a reliable alternative in finite-state, finite-action scenarios. However, in a continuous state-action space, the uncertain sets of transition probability matrices cannot be described as above. Another drawback of the alternative that Robust Reinforcement Learning offers is that the policy that will be obtained will be much more conservative than the policy that could be obtained using alternative methods as it will account for the uncertainty in the transitions when minimizing the expected costs [35]. The policy obtained using robust RL may have a poor performance even in cases where no uncertainty is involved due to its nature. It is for these two reasons, the scalability of the state-action space and the robustness-performance trade-off, that robust RL may be considered as an extension to make the proposed algorithm robust to perturbations. Nevertheless, no further research will be performed trying to derive the algorithm from robust control theory directly.

4-2 Transfer Learning

In 1976 the idea of Transfer Learning was introduced by Stevo Bozinovski and Ante Fulgosi [36] in neural networks training. The main goal of Transfer Learning is storing prior knowledge about a problem and applying it to similar applications. In 2016 during the NIPS conference, Andrew Ng pointed out that Transfer Learning could be the next driver of Machine Learning success.

Transfer Learning has a significant practical value as it helps to reduce significantly the sample-complexity, making data collection less expensive. In 2009 a survey was written by M. Taylor [37] collecting the knowledge about Transfer Learning published until then in the field of Reinforcement Learning. Transfer Learning work has been focused on transferring policies, control parameters or dynamics.

This method is widely used in Deep Neural Networks that have been already trained. By reusing learned feature vectors for a new application, they can be adapted making the training of the new neural network more efficient. It is also very useful when various agents are involved in the same control process [38] and transferring information between them help obtain the optimal policy faster than if every agent would not have access to the information.

4-3 System Identification

The last approach that is explored in Literature is System Identification. In the case where prior information about the environment is known, we can include the information in an offline

System Identification process to obtain estimates of the system transitions and then perform model-based methods with the parameters learned. However, a more interesting approach, Online System Identification (OSI), was discussed in 2017 by W. Yu [3] in combination with model-free algorithms. The algorithm will be trained to predict the unknown parameters and then they will be incorporated into the policy along with the current state.

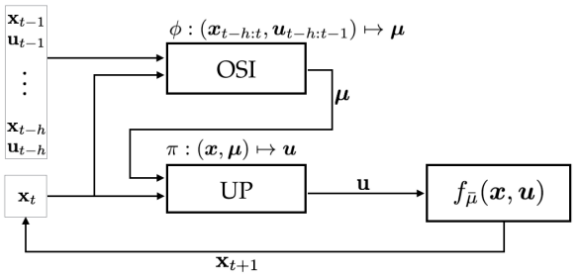


Figure 4-2: The Online System Identification provides a policy π to the controller that provides the action to the environment [3]

The Online System Identification block will be implemented as a supervised learning problem [39] that will be trained online. $\phi : (\mathbf{x}_{t-h:t}, \mathbf{u}_{t-h:t-1}) \mapsto \boldsymbol{\mu}$ will predict the dynamic parameters μ given the current state and the history of past state-action pairs. OSI will be implemented using a standard linear neural network. However, the results in Literature for fairly simple environments show that to train an accurate model a lot of samples are required. As one of the goals of the thesis will be to implement a sample-efficient algorithm this alternative will be discarded.

Chapter 5

Mixed Iterations

The algorithm that will be discussed as a solution to the problem posed will be called Mixed Iterations (MI). The aim of the algorithm will be to fill the gap in the literature for the cases where partial information is known about the system dynamics improving the convergence against the model-free algorithms. The improvement will lead to a better convergence with fewer samples. The algorithm will be impactful in applications where every iteration has an associated cost. The algorithm will be aimed to solve MDPs with a transition probability tensor $\mathbb{P}$.

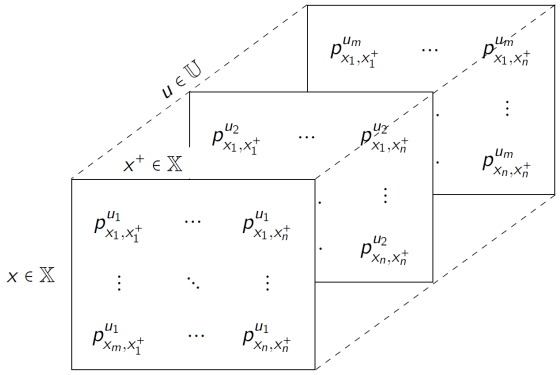


Figure 5-1: Transition Probability Tensor $\mathbb{P}$ of MDP with full information

The idea behind the algorithm will be to check the amount of previously known transitions for a certain state and perform a weighted average of model-based and mode-free iterations to update the cost function. In the figure above it can be observed that every transition between a state x and a future state x^+ given an action u is determined by a probability p^u_{x,x^+} that can be known or not. The algorithm will update the cost function $Q(x,u)$ considering the known transitions assigning them their corresponding probability and assigning the rest of the unknown probability to an update similar to the one in Q-Learning where a future state will be sampled from the environment.

To this end, consider a discrete random variable $Y \in \{y_i\}_{i\in\mathbb{J}}$, with a partially known probability mass function $p_i = \mathbb{P}(Y = y_i)$ for $i \in \mathbb{I} \subset \mathbb{J}$. We will use the following approximate expectation

$$\mathbb{E}(y) \approx \sum_{i\in\mathbb{I}} p_i y_i + \left(1 - \sum_{i\in\mathbb{I}} p_i\right) \hat{y} \tag{5-1}$$

where $\hat{y}$ is the sample from the system. The approximation of the expectation of the variable Y, assumes that $p_i > 0$ for all $i \in \mathbb{I}$ and $\sum_{i\in\mathbb{I}} p_i \leq 1$. We now use the approximation above to describe a single mixed iteration. For each $(x, u) \in \mathbb{X} \times \mathbb{U}$, let $\mathbb{Y}_{x,u} \subset \mathbb{X}$ be the set of next states x^+ for which $\mathbb{P}(x^+|x, u) > 0$ is known. Now, for a sample $\hat{x}^+$ of $x^+ \sim \mathbb{P}(\cdot|x, u)$, define

$$\tilde{\mathcal{T}}Q(x, u, \hat{x}^+) = \sum_{x^+\in\mathbb{Y}_{x,u}} \mathbb{P}(x^+|x, u)\hat{\mathcal{T}}Q(x, u, x^+) + \left(1 - \sum_{x^+\in\mathbb{Y}_{x,u}} \mathbb{P}(x^+|x, u)\right) \hat{\mathcal{T}}Q(x, u, \hat{x}^+) \tag{5-2}$$

where the sampled Bellman operator is defined as

$$\hat{\mathcal{T}}Q(x, u, y) = c(x, u) + \gamma \min_{u\in\mathbb{U}} Q(y, u). \tag{5-3}$$

The new value in $Q_{k+1}(x, u)$ will be computed using the two-step update it was used in Q-Learning to help reduce the influence of sampling choosing an appropriate step size α_k,

$$Q_{k+1}(x, u) = Q_k(x, u) + \alpha_k \left(\tilde{\mathcal{T}}Q_k(x, u, \hat{x}^+) - Q_k(x, u)\right). \tag{5-4}$$

Observe that $\tilde{\mathcal{T}}$, similar to $\hat{\mathcal{T}}$, is a consistent estimator of the true Bellman operator $\mathcal{T}$ and the mixed iteration (5-2) also converges to the optimal Q-function as longs as all the state-action pairs with partially unknown transition probabilities (i.e., with $\sum_{x^+\in\mathbb{Y}_{x,u}} \mathbb{P}(x^+|x, u) < 1$) are visited infinitely often. Also, notice that the newly proposed update rule (5-4) will have a higher computational cost per iteration compared to Q-Learning, since it needs extra computations corresponding to the known transitions. To be precise, if the size of $\mathbb{Y}_{x,y}$ is upper-bounded by ℓ for all $(x, u) \in \mathbb{X}\times\mathbb{Y}$, then the per-iteration complexity of Mixed Iteration algorithm is of $O(nm^2(\ell))$ which in the worst case can be $O(n^2m^2)$. As the per-iteration complexity will increase with the number of states and actions, it can be predicted that despite the algorithm will potentially reduce the number of iterations needed for convergence in large state-action spaces the computational time required will be higher than Q-Learning. However, it also needs to be pointed out that despite the per-iteration complexity being worse than Q-Learning, in the case of a bigger state space, the actual number of iterations can be cut down as not all the states are reachable from a certain position as most of the entries of the probability tensor will be zero. Actually, the number of future states in the cases studied is proportional to the number of actions in the environment so the real per-iteration complexity will be closer to $\mathcal{O}(nm^3)$ rather than $\mathcal{O}(n^2m^2)$. In the benchmark environments where the algorithm will be tested, this makes a huge difference as the number of actions is significantly less than the number of states. Notice that the per-iteration complexity refers to the number of mathematical operations needed to complete one iteration, while the iteration complexity will be the number of iterations required to converge. The total computational complexity will be the combination of both and the success in the performance of the designed algorithm will be evaluated in terms of the computational complexity of the designed algorithm compared to Q-Learning.

5-1 The Mixed Iterations Algorithm

The algorithm shown next reflects how the implementation of the algorithm will look in the case where a generative model can be used and how we define the convergence of the algorithm within the threshold ϵ. The inputs required for the algorithm will be explained as follows. The first variable ϵ will be the convergence threshold that will determine when the error between two consecutive iterations is close enough, $\mathbb{X}$ and $\mathbb{U}$ will be the possible set of states and actions that comprise the state-action space, then $\mathbb{Y}_{x,u}$ the set of next states x^+ for which $\mathbb{P}(x^+|x,u) > 0$ is known, $c(x,u)$ determines the stage cost of choosing a particular action being at a particular state, $\mathbb{P}$ is the transition probability tensor that defines the dynamics of the MDP, γ is the discount factor between 0 and 1, and finally, k_{max} will be the maximum number of iterations the algorithm will be allowed to run due to the potential high number of iterations required to converge.

Algorithm 3 Mixed Iterations

Input: ϵ, $\mathbb{X}$, $\mathbb{U}$, $\mathbb{Y}_{x,u}$, $c(x,u)$, γ, k_{max}
Output: π, the ϵ-optimal policy

```
Q_0(x,u) ← 0_{n×m}
while max_{(x,u)} |Q_k(x,u) − Q_{k+1}(x,u)| > ε and k < k_max do
    for x ∈ 𝕏, u ∈ 𝕌 do
        x̂⁺ ∼ ℙ(· | x, u)
        T̃Q(x,u,x̂⁺) = Σ_{x⁺∈𝕐_{x,u}} ℙ(x⁺|x,u) T̂Q(x,u,x⁺) + (1 − Σ_{x⁺∈𝕐_{x,u}} ℙ(x⁺|x,u)) T̂Q(x,u,x̂⁺)
        Q_{k+1}(x,u) = Q_k(x,u) + α_k (T̃Q_k(x,u,x̂⁺) − Q_k(x,u))
    end for
end while
π(x) = arg min_{u∈𝕌} Q(x,u)
```

Chapter 6

Design of the Experiments and Simulation Results

In order to test the algorithm a number of experiments were designed in increasing order of complexity. The experiments will analyze the performance of the algorithm with various amounts of previous knowledge depending on the percentage of known transitions: 0% (Q-Learning), 20%, 40%, 60%, 80% and 100% (Value Iteration). The percentage of known transitions determines the sum of known transitions; for example, by 20% knowledge, we mean that the known probabilities $\mathbb{P}(x^+|x,u)$ add up to 20% of all possible transitions $(x,u) \to x^+$. If we recall the definition of $\mathbb{Y}_{x,u}$ as the set of next states x^+ for which $\mathbb{P}(x^+|x,u) > 0$ is known, then the 20% knowledge is defined as

$$\sum_{(x,u)\in\mathbb{X}\times\mathbb{U}} \sum_{x^+\in\mathbb{Y}_{x,u}} \mathbb{P}(x^+|x,u) = \frac{20}{100} \times |\mathbb{X}|| \cdot |\mathbb{U}|. \tag{6-1}$$

The percentage of previously known transitions will remain the same during the different runs, however, the unknown transitions will be randomized so the known transitions are different between runs. The results will be averaged between runs so there will be different combinations of known transitions. In the MDPs that will be used as benchmarks, there will be key transitions between states that if unknown can badly influence the behaviour of the proposed algorithm. The same can happen the other way around leading to overoptimistic results. So along the iterations that will comprise one run the known transitions will remain constant, but between runs, they will change to obtain results that are fair and representative of the performance of the algorithm.

There will be different environments where the algorithm will be tested. These environments will be the `RiverSwim` MDP, a varying size GridWorld and two standard problems used as a benchmark in RL algorithms, the MountainCar and the Inverted Pendulum.

- `RiverSwim` MDP: six states, structured rewards and transitions, two actions per state.

- GridWorld: varying size grid world 7x7 and 11x11 with obstacles and walls. Structured rewards and five actions per state.
- The Mountain Car: a discretized problem where the car wants to reach the top of a mountain with three possible actions.
- Inverted Pendulum: a discretized problem where the pendulum needs to be stabilized in the unstable upwards equilibrium point with three possible actions to be taken by the controller.

In the last three scenarios, without noise, the outcome of the future states would be deterministic, and for this reason, it was decided that noise in the actions would be included to make the outcome of an action stochastic. The same kind of noise will be included across the three scenarios. With an 80% probability the chosen action will be the one that will be performed, then there will be a 10% chance where there will not be any action performed and finally, there will be a 10% where the action performed will be any other action chosen randomly with uniform distribution.

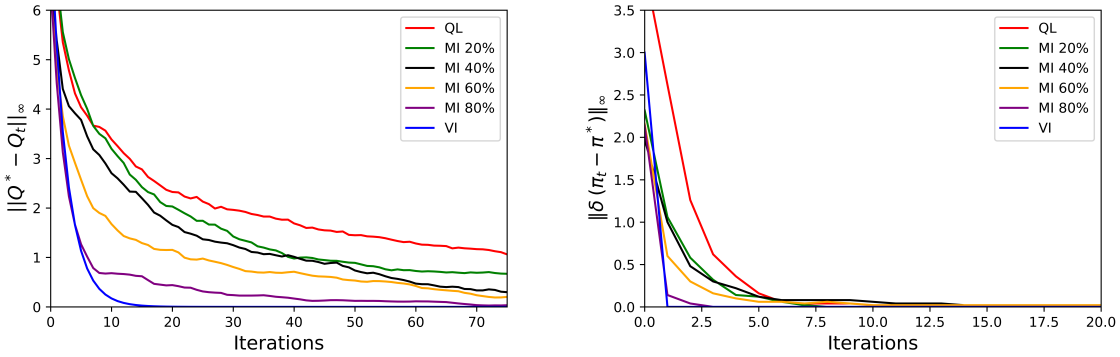
There is a last point that needs to be discussed, the metrics that will be used to determine the success of the algorithm. The idea of an RL algorithm is to provide the optimal policy to follow at every time step. For this reason, the metric that will be used to evaluate the performance of the algorithm will be the difference between greedy policy π_k at each iteration and the optimal policy π^* computed as the number of states $x \in \mathbb{X}$ for which $\pi_k(x)$ is not the optimal decision, that is, $\pi_k(x) \neq \pi^*(x)$. We use $\delta(\pi_k - \pi^*)$ to denote this difference. When tracking the evolution of $Q_k(x,u)$ to $Q^*(x,u)$ in larger state-action spaces it was soon realized that the number of iterations required for a clear convergence was way more than in the case of just obtaining the optimal policy. For this reason the evolution of $Q_k(x,u)$ will be computed in the first two scenarios but in the case of more complex state spaces, the results will be limited to comparing the optimal policy to the one obtained with the greedy policy obtained applying the Mixed Iteration algorithm. When using Q-Learning, the controller will learn the optimal cost of every state-action pair through sampling assuming that we have a generative model that can generate the future state $x^+ \sim \mathbb{P}(\cdot|x,u)$.

6-1 Riverswim MDP

The first case where the algorithm will be implemented will be the simplest out of all of them and it will be the Riverswim MDP. In this MDP that was already introduced before in Figure 3-1 there are six states and the goal is to get to the terminal state and remain there to maximize the reward. There will be two possible actions, left or right and the outcome of choosing these actions will be determined by the probabilities determined by the model of the RiverSwim MDP. The rewards are structured assigning a low negative value to every state apart from the terminal one, and then a positive reward for reaching the terminal state.

In order to evaluate the performance of the algorithm the two metrics discussed at the beginning of the section will be obtained, the error of Q_k with respect to the optimal action-value function Q^* and the error of the corresponding greedy policy $\delta(\pi_k - \pi^*)$. As in this example the number of states is not significant the norm of the error of $Q_k(x,u)$ can be obtained and the convergence is obtained after a few iterations.

The probability kernel will be different in every run taking into account 50 different combinations of known probability tensors. For every run, the probability kernel will remain the same and across all runs the sum of the known transitions will be equal to the percentage of known transitions that we are trying to study, 20%, 40%, 60% or 80% as it was shown in Equation 6-1.



(a) Evolution of the average error of Q_k respect Q^* over 50 runs.

(b) Evolution of the average number of states x with for which $\pi_k(x) \neq \pi^*(x)$ over 50 runs.

Figure 6-1: Performance of Mixed Iteration (MI) Algorithm with varying percentage of known transitions in comparison with Q-leaning (QL) and Value Iteration (VI) algorithms for the River-Swim MDP.

The first results obtained after applying the algorithm to a simple MDP show that the algorithm behaves as intended. The limit imposed by the convergence of Q-Learning is constantly respected and depending on the percentage of known transitions the performance of the algorithm is closer to Value Iteration. However, in this case, the gap between Value Iteration and the Mixed Iteration algorithm is still significant due to the structure of the MDP and having just a few probabilities that describe the evolution of the process. When some of these few probabilities are unknown the Mixed Algorithm resembles more the Q-Learning behaviour than the Value Iteration algorithm because the algorithm needs to sample more of these unknown transitions. The improvement with respect to the optimal Q^* is not very significant compared to Q-Learning because of the reasons we just presented. If instead the policy obtained at every time step is evaluated, it can be observed that the convergence is quicker in all the scenarios where partial information is known; this is because the policy just needs to know which action reward is higher and pick the optimal action without having to wait until Q_k converges. From now on, the metric that will be used to evaluate the performance of the algorithm will be the latter one as it offers interesting insights into the behaviour of the algorithm without having to compute large amounts of iterations.

6-2 GridWorld

The second scenario where the algorithm has been tested is in different sizes of GridWorlds. Testing the algorithm in this kind of scenarios allows us to increase the size of the state-action space significantly and evaluate the possible scalability of the algorithm.

In the GridWorld the rewards are structured so that every step is penalized with a small and a big negative reward when the terminal state is reached. For every state, 5 possible actions can be performed (north, south, east, west, stay). To add some stochasticity to the process at every time step there will be a 10% chance of taking a random action instead of the chosen one and a 10% chance of the agent not moving. The initial position will be randomized so every run is different and the optimal policy is achieved for every state. In the images below the mazes used are shown, the darkest shade orange represent lava states that will be heavily penalized, the terminal state is the bottom-right corner represented in brown. The environments have been obtained with a random maze generator.

Figure 6-2: Shape of the different size mazes used to test the algorithm

For the two mazes, we will use the same metrics used in the previous MDP. However, due to a larger state space, the number of different runs will be reduced due to computational time restrictions.

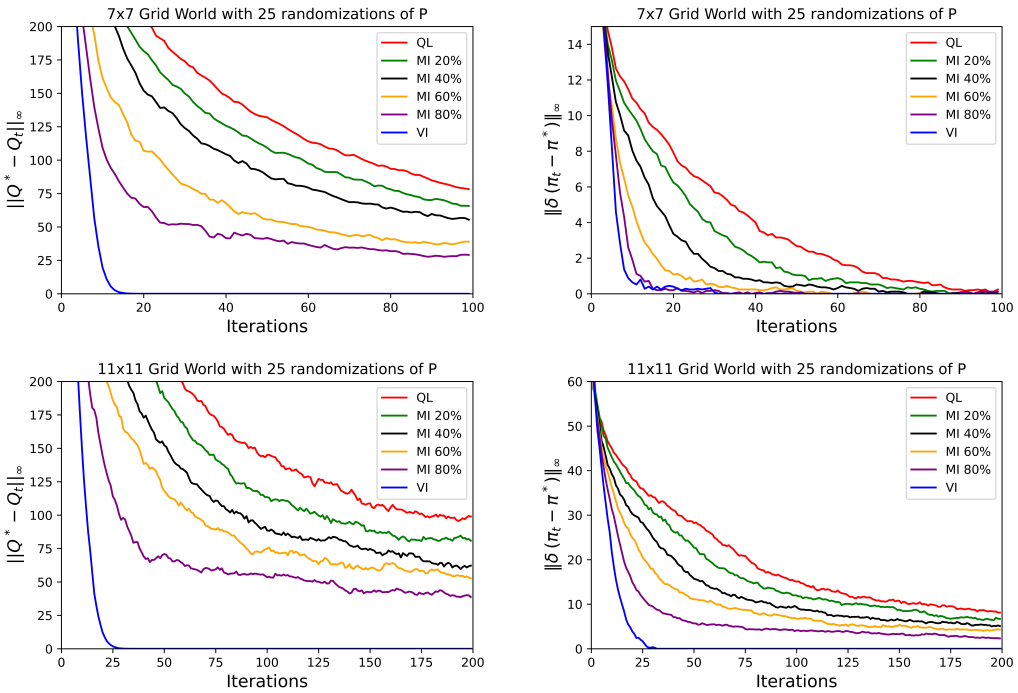


Figure 6-3: Average evolution of the error of Q_k and the number of elements in the policy $\pi_k(x) \neq \pi^*(x)$ using Q-Learning (QL), Value Iteration (VI) and Mixed Iterations (MI) over 25 runs

The results obtained after applying the Mixed Iteration algorithm compared to Q-Learning and Value Iteration are satisfactory. The algorithm behaves as was expected and the iterations needed for convergence towards the optimal policy are significantly less than in Q-Learning. However, a trend can be observed comparing the results obtained in this example and in the previous environment. The smaller the state-space is, the bigger the improvement is compared to Q-Learning. The behaviour of Q-Learning and the Mixed Iterations algorithm partially depends on sampling the states, thus if the state-space is larger, more sampling needs to be performed to obtain an accurate cost function. It can also be seen that the cost functions Q_k slowly converge to the optimal Q^* while the policy converges way faster even in larger state spaces.

6-3 The Mountain Car Problem

In this application of the algorithm proposed in this section, we will consider the challenge of driving an underpowered car until the top of a steep mountain road. The main problem is that gravity will be stronger than the power of the car's engine, so even if we fully accelerate, the car cannot reach the top of the hill. The solution to this problem relays on moving in the opposite direction from the goal to build inertia up so the car can reach the top of the hill. This example serves as a toy example of a control environment where the controller has to learn the counterintuitive dynamics of the system to achieve the completion of the task.

The stage reward will be defined as -1 for every state that is not the goal. When the goal is reached, a reward of +1 will be given. In this problem set-up three possible actions u_k are considered, full throttle forward, full throttle backwards and zero throttle. In the deterministic case of the Mountain Car problem, two observable states of the system x_t and v_t will evolve according to the simplified dynamics,

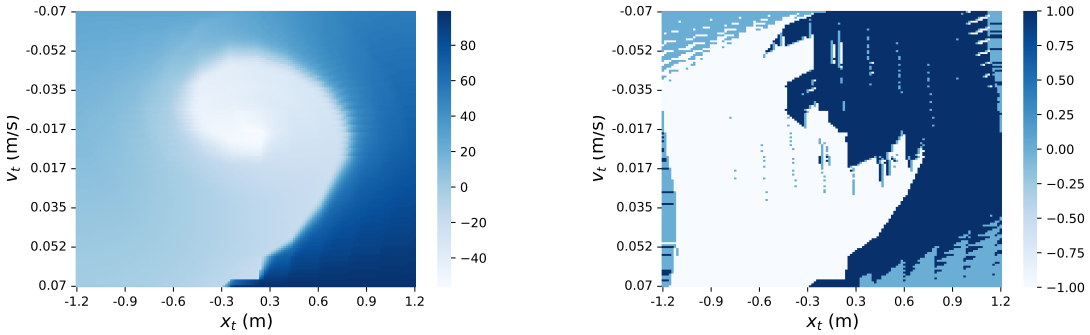
$$\begin{aligned} v_{t+1} &= v_t + 0.001u_t - 0.0025\cos(3x_t) \\ x_{t+1} &= x_t + v_{t+1} \end{aligned} \tag{6-2}$$

where bounds on both states will be placed ensuring that $-1.2 \leq x_t \leq 0.5$ and $-0.07 \leq v_{t+1} \leq 0.07$. It will be assumed that the left wall produces an inelastic bound, so in the case of x_{t+1} being -1.2 then $\dot{x}_{t+1}$ will be reset to 0. The initial position of the car will be a random position $x_0 \in [-0.6, -0.4]$ and $v_0 = 0$.

In the studied case, the dynamics will be considered as described before, however, the choice of action will not be deterministic. It will be considered that in the case of choosing a particular action, there is an 80% chance of ending up in the desired state, a 10% chance of not performing any action and a 10% chance of performing the opposite action. These probabilities will be implemented in a transition probability kernel $\mathbb{P}$ that will be later used to describe the amount of knowledge of the system dynamics.

The goal of the problem is to find an optimal policy for every state so the final position is achieved and collect the reward. To solve the problem we will use the same two algorithms used before, Value Iteration and Q-Learning. When using the Value Iteration algorithm the controller will know the dynamics of the system and the probability of ending up in a state that was not the desired one chosen by the policy. The two algorithms mentioned before will be used as benchmark and the Mixed Iteration algorithm will be tested with previous

knowledge of 25%, 50% and 75% of the transitions between states. Our algorithm will be applied to a discretized version of a continuous state environment. We will first obtain the optimal value function as well as the optimal policy using Value Iteration, the results obtained are shown in Figure 6-7. Once they were obtained we can compare the algorithm's evolution along the iterations until it converges.



(a) Optimal cost function J^* **(b)** Optimal policy π^*

Figure 6-4: Optimal cost and optimal policy obtained using model-based methods for Mountain Car

The results obtained are expected to be noisier than in the previous cases as, due to the large size of the state space, it was not possible to run a large number of runs and then average them out and obtain statistical results about the evolution of the policy. We compared the evolution of the policy along the iterations using the Mixed Iterations algorithm to see if an improvement with respect to Q-Learning can be obtained, in this case with a much bigger state-action space. Figure 6-5 shows the evolution of the elements of the policy π_k with respect to the optimal policy π^*. As shown, we can see an improvement with respect to Q-Learning, however, the improvement was not as significant as in previous examples. Due to the size of the state-action space, the number of different runs was just three, and the plot shows the average between these three runs.

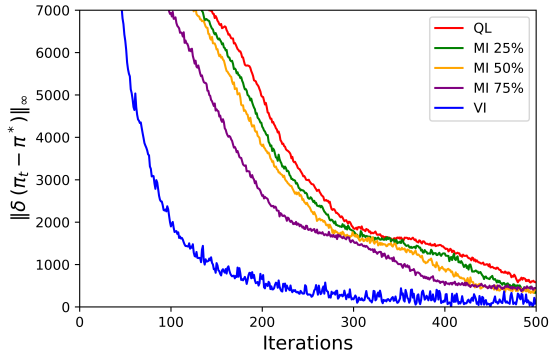
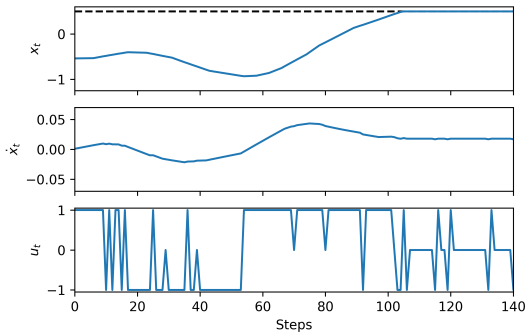
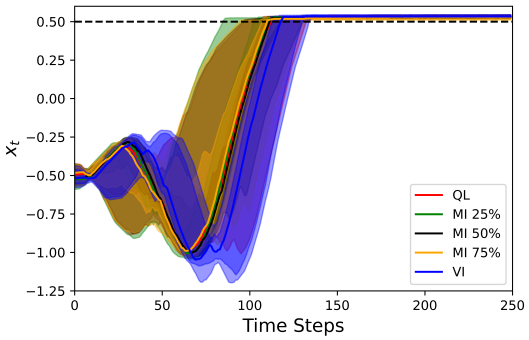


Figure 6-5: Average of the evolution of the error of $\pi(x)$ along iterations with Value Iteration (VI), Q-Learning (QL) and Mixed Iterations (MI) algorithms over three runs

After obtaining the policies for every case, we selected the one with 75% prior knowledge and tested them simulating the dynamics of the Mountain Car problem. To do so, the policy was used to generate the actions and it can be easily observed that the goal of reaching the final state $x = 0.5$ was reached after a couple of swings in the bottom part of the mountain to get enough inertia so it could get to the top. The sequence of actions was also plotted to show the noise in the actions showing the non-optimal actions taken during the control sequence. What one could expect is that in Figure 6-6a the optimal action sequence will be first 1 then -1 and finally 1 again until the goal is reached. However, Figure 6-6a shows the amount of non-optimal actions chosen. Also, a series of control sequences were performed with different initial conditions to show that the obtained policy is able to control the car across the interval of possible initial conditions. In a thicker line, we can see the average of the trajectories with varying initial conditions proving that the obtained policy can successfully control and solve the problem.



(a) States and Actions during a control sequence



(b) Evolution of the position x_k over a control sequence

Figure 6-6: Evolution of the states during a sequence of control actions over different initial conditions

We can conclude that the learning was completed successfully and the controller can drive the car to the top of the mountain. It is an important step as these results open the door for possible implementations of the algorithm to classical control problems with discretized dynamics.

6-4 The Inverted Pendulum Problem

The next example where the algorithm will be tested is the inverted pendulum problem. The goal of the algorithm will be to obtain a policy to keep a pendulum in the upright position, as the upright position is an unstable equilibrium point the controller needs to continuously provide control actions to remain in the upright position. The discrete-time dynamics of the pendulum are defined as shown in the equations below,

$$\phi = \frac{1}{M+m} \tag{6-3}$$

$$\alpha_{t+1} = \frac{g\sin(\theta_t) - \phi m \omega_t^2 \frac{\sin(2\theta_t)}{2} - \phi\cos(\theta_t)u_t}{\frac{4}{3}l - \phi\cos^2(\theta_t)} \tag{6-4}$$

$$\omega_{t+1} = \omega_t + \alpha_{t+1} \tag{6-5}$$

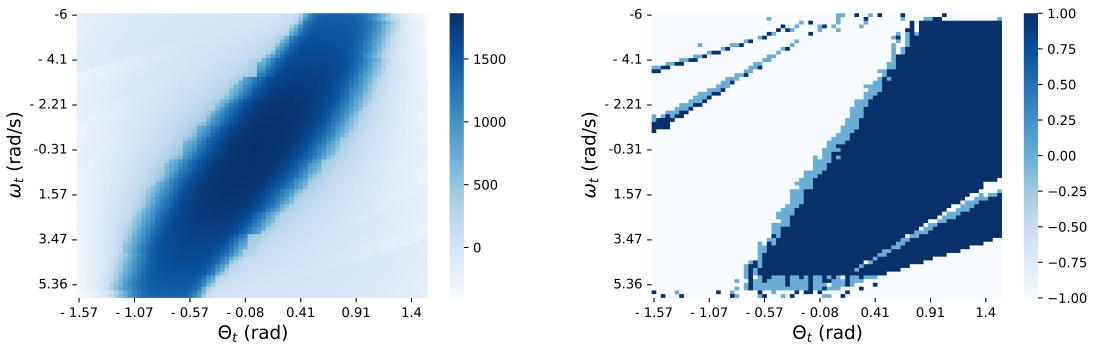
$$\theta_{t+1} = \theta_t + \omega_{t+1}. \tag{6-6}$$

The upright position will be set as the reference with $\theta = 0$ rad, the states will be bounded ensuring that $-\frac{\pi}{2} \leq \theta_t \leq \frac{\pi}{2}$ and $-6 \leq \omega_t \leq 6$ and also the initial conditions will be set to $\theta_0 \in [-0.7\,,\,0.7]$ rad and $\omega_0 = 0$ rad/s. The controller can perform one of three actions u_t and apply a +50N, 0N or -50N force to the base of the pendulum. The stage reward is designed to penalize the two observable states, the angular position θ and the angular velocity ω. For this reason, the value function will be designed as follows and the variables k_1, k_2 and k_3 will be tuned to optimize the performance of the algorithm using the least amount of energy used will also be considered,

$$c((\theta, \omega), u) = k_1 \cos(\theta) - k_2\omega^2 - k_3u^2 \tag{6-7}$$

The goal of the problem will be to obtain the optimal policy to control the pendulum. However, the choice of action will not be deterministic as there will be a stochasticity added to the control process. As done in previous experiments, there will be an 80% chance to follow the chosen action, a 10% chance of not doing anything and a 10% chance of following an action other than the chosen one. This stochasticity will be implemented in the probability transition tensor $\mathbb{P}$ that will describe the dynamics of the MDP. The percentage of probabilities known will vary depending on the different runs ranging from 0% in the case of Q-Learning to 100% in the case of Value Iteration. All the percentages in between will be solved using the Mixed Iterations algorithm and the improvement will be compared to the Q-Learning algorithm.

Before testing the algorithms, the optimal value function, as well as the optimal policy, will be computed offline to see the complexity of these two. These are computed via the Value Iteration algorithm with full knowledge of the dynamics of the system and the added stochasticity. The obtained optimal value function and policy are shown in Figure 6-7 and in particular, we can see a simple structure in the optimal policy. It is expected that as opposed to the Mountain Car problem where the optimal policy was quite complex, in this problem the algorithm will achieve better results in terms of the number of iterations needed for convergence.



(a) Optimal cost function J^* **(b)** Optimal policy π^*

Figure 6-7: Optimal cost and optimal policy obtained using model-based methods for the Inverted Pendulum

After the policy and value function are obtained, the main goal of the experiment is to implement the Mixed Iteration algorithm with various amounts of prior knowledge. As the state-action space is significantly large, the amount of partial previous knowledge will be considered to be 25%, 50% and 75%. Figure 6-8 shows the performance of Mixed Iteration, averaged over five runs, in comparison with Q-Learning and Value Iteration.

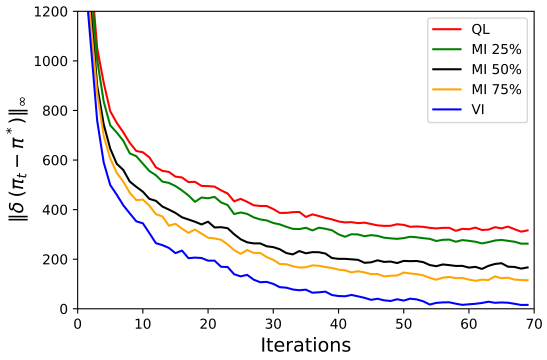
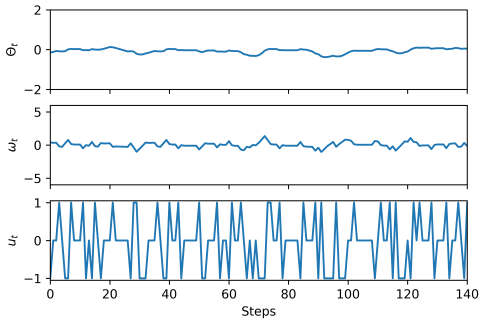
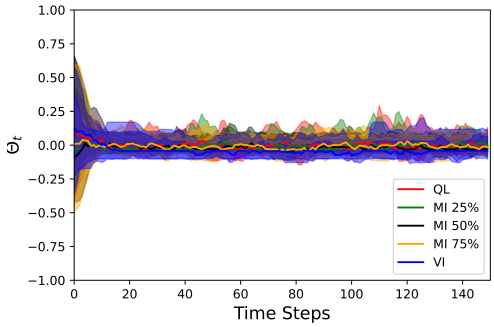


Figure 6-8: Evolution of the error of $\pi_k(x)$ along iterations with Value Iteration , Q-Learning and Mixed Iteration algorithms

As it was predicted, the Mixed Iteration algorithm improves the performance with respect to Q-Learning proportionally depending on the amount of prior knowledge. After the 70 iterations, a policy was obtained for each algorithm. Figure 6-9b compares the performance of these policies in keeping the pendulum in upward position for an interval of different initial conditions. To show the performance of these policies obtained we will plot the evolution of the states and a sequence of control actions in Figure 6-9a with the obtained policy obtained with the Mixed Iteration algorithm with 75% previous knowledge. The two states remain within the expected values and try to keep the pendulum close to the equilibrium point and with angular velocity close to 0 rad/s. In the right figure the evolution of the angular position is shown under different policies.



(a) States and Actions during a control sequence



(b) Evolution of the position θ_k over a control sequence

Figure 6-9: Evolution of the states during a sequence of control actions over different initial conditions

It can be therefore concluded that the algorithm implemented is working successfully as

it meets the expected performance improving the number of iterations with respect to Q-Learning and performing as expected despite the different kinds of randomizations in the probability transition tensor.

6-5 Summary of the Results

In this section, we will compile and compare the data collected from the various sub-sections of the study. The comparison will focus on evaluating the performance of the algorithm in relation to its requirements. A table will be provided that displays the number of iterations required to achieve a 90% accuracy in the policy, as well as the improvement percentage in the iteration complexity with respect to Q-Learning. To calculate the improvement percentage with respect to Q-Learning, Value Iteration was used as a reference point. A 100% improvement indicates that the algorithm performs similarly to Value Iteration before reaching 90% accuracy in the policy. Therefore, we will not present any improvement percentages for Value Iteration and Q-Learning, as we will be testing the Mixed Iteration algorithm with various levels of prior knowledge in comparison to these well-established algorithms.

	RiverSwim MDP		***GridWorld 7x7***		***GridWorld 11x11***	
	Iterations	*Percentage of Improvement*	*Iterations*	*Percentage of Improvement*	*Iterations*	*Percentage of Improvement*
Q-Learning	5	-	42	-	126	-
Mixed Iterations 20%	4	25%	31	30.56%	101	21.62%
Mixed Iterations 40%	3	50%	20	61.11%	72	50.45%
Mixed Iterations 60%	2	75%	13	80.56%	48	70.27%
Mixed Iterations 80%	1	100%	10	88.89%	26	91.89%
Value Iteration	1	-	6	-	15	-

	Mountain Car		***Inverted Pendulum***	
	Iterations	*Percentage of Improvement*	*Iterations*	*Percentage of Improvement*
Q-Learning	298	-	53	-
Mixed Iterations 25%	282	8.12%	33	53.85%
Mixed Iterations 50%	271	18.78%	24	76.92%
Mixed Iterations 75%	235	31.98%	18	92.31%
Value Iteration	101	-	15	-

Table 6-1: Summary of the results obtained after applying the Mixed Iterations algorithm to the different environments proposed, the number of iterations to achieve a 90% accuracy and the improvement respect Q-Learning are shown

In the table above, the improvement that was already observed with the plots obtained across this section can be quantified. If we look at the requirements defined when the algorithm was designed, the algorithm is able to converge within the same conditions as Q-Learning and Value Iteration. Also, the algorithm converges under any percentage of known transitions. It can also be derived from the results obtained that the algorithm is not suitable when we have little information about the transitions defining the dynamics of the MDP. As the algorithm derived has a higher per-iteration complexity than Q-Learning, we can conclude

that when more than 50% of the transitions are known this could be a good alternative as the improvement is significant. For fewer amounts of prior knowledge, we would discourage using this algorithm and look for other possible approaches.

Chapter 7

Final Remarks

In the following Section 7-1, we will evaluate the performance of the proposed Mixed Iterations algorithm in comparison to the base algorithm that should have been used in a model-free scenario, Q-Learning. Through this comparison, we will gain a deeper understanding of the strengths and weaknesses of the algorithm proposed. Then, we will examine the proposed algorithm and its limitations in Section 7-2. Additionally, we will explore potential directions for future work in Section 7-3, to improve upon the algorithm and address the identified limitations. Furthermore, the analysis performed will provide insight into the field of reinforcement learning and model-free alternatives as well as the challenges that researchers in this will field continue to face.

7-1 Performance of the algorithm in comparison to Q-Learning

The main question posed in the thesis was whether it is possible to improve the iteration complexity of an MDP with partial knowledge about the transitions compared to using model-free algorithms like Q-Learning. To do so, the results obtained for the number of iterations required to achieve a 90% accuracy in the policy will be used in combination with the per-iteration complexity derived for both algorithms in Chapters 4 and 5. After obtaining a clear conclusion about when it is better to use the proposed algorithm, we can recommend its use in certain scenarios and discard this approach in others, finding better alternatives than the Mixed Iteration algorithm.

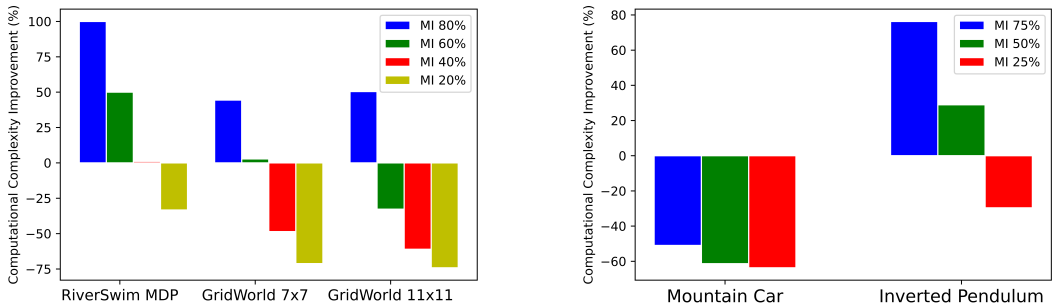
A distinction needs to be made between the per-iteration complexity and the iteration complexity. The per-iteration complexity refers to the number of mathematical operations needed to complete one iteration, while the iteration complexity will be the number of iterations required to achieve the 90% accuracy mentioned. The total computational complexity will be the product of these two complexities and will allow us to assess the performance of the algorithm and to conclude if it saves computational time compared to Q-Learning. The iteration complexities of Q-Learning and Mixed Iterations are $\mathcal{O}(m^2n)$ and $\mathcal{O}(m^3n)$, respectively. In

Table 7-1, the number of states and actions for each test environment and the corresponding iteration complexities are provided.

	States (n)	Actions (m)	Per-Iteration Complexity	
			$\mathcal{O}(nm^2)$	$\mathcal{O}(nm^3)$
RiverSwim MDP	5	2	20	40
GridWorld 7x7	49	5	1225	6125
GridWorld 11x11	121	5	3025	15125
Mountain Car	19600	3	176400	529200
Inverted Pendulum	5625	3	50625	151875

Table 7-1: State-Action spaces dimensions and Iteration Complexities across test environments

Once we have the number of iterations required for a 90% accuracy, the size of the state-action space and the per-iteration complexity for both Q-Learning and Mixed Iteration, a fair comparison can be made across all different test environments with varying amounts of prior knowledge. To make this comparison, we will evaluate the percentage of improvement or decrease in performance of the Mixed Iteration algorithm with the Q-Learning algorithm. To do so, the obtained per-iteration complexity will be multiplied by the number of iterations obtained in Section 6 and based on these results we can compute the percentage of improvement with respect to Q-Learning. In the case of resulting in negative percentages, we can conclude that the algorithm performs worse than Q-Learning in terms of computational complexity, the same logic applies otherwise. In Figure 7-1, we can see the improvements achieved for all different percentages of known transitions in all five environments. The idea behind plotting the computational complexity improvement in these bar plots is that conclusions can be obtained at first sight.



(a) Improvement of MI20%, MI40%, MI60% and MI80% in RiverSwim MDP and Grid World

(b) Improvement of MI25%, MI50% and MI75 in Mountain Car and Inverted Pendulum

Figure 7-1: Percentage of Improvement of Mixed Iterations respect to Q-Learning in all environments

In the obtained results, we can see that the computational complexity is not always improved compared to Q-Learning. Despite the plots that show the evolution of the correct number of elements in the policy pointing towards promising results, the Mixed Iteration algorithm has a higher per-iteration complexity. As a result, the use of the Mixed Iterations algorithm

can only be recommended when the previous amount of known transitions is large enough to compensate for the per-iteration complexity. It should also be noted that in the case of the Mountain Car problem, the results are more disappointing than in the other scenarios. This could be due to the complex shape of the policy and the algorithm's inability to explore the state space efficiently enough with the given number of iterations. However, despite the Mountain Car being an outlier among the test environments chosen, the results show that the use of the proposed algorithm would be suitable only when the amount of previously known transitions is above 75%. When this condition is met, the improvement with respect to Q-Learning is significant in 4 out of the 5 environments. For other percentages of known transitions between 60%-40%, an in advance examination of the complexity of the policy would be recommended and, if possible, a simulation test to see how it would perform. In cases where less than 40% of the transitions are known, we would discourage using this algorithm and proceed with Q-Learning.

7-2 Limitations of the algorithm

In this section, the limitations of the proposed algorithm will be discussed. We also identify areas where it may not perform better than Q-Learning based on the results obtained. As it was concluded in the previous section, the algorithm is not suitable across all scenarios. We would like to discuss the scenarios where the algorithm does not perform better than Q-Learning, provide reasons why it does not work and lately establish clear limitations on the success of the algorithm.

Three main limitations were found related to the performance of the algorithm: the first is sensitivity to the size of the action space, the second is difficulty in obtaining policies in unlikely states, and the third is the assumption that a generative model can generate future states with the correct probabilities. This third limitation is due to the assumption that an available generative model is present, which in some real-world cases may be impossible to know.

Dimension of the Action Space The goal of the proposed algorithm is to reduce the number of iterations and computational operations needed for convergence compared to Q-Learning, making the process more sample efficient. However, there is one clear limitation that needs to be addressed. The Mixed Iteration algorithm has a larger per-iteration complexity, $\mathcal{O}(nm^3)$, compared to Q-Learning, $\mathcal{O}(nm^2)$. For this reason, depending on the size of the action space, the algorithm will need to have a higher percentage of improvement to compensate for the larger amount of operations needed to complete one iteration. In most RL problems, this is not a problem as large state-action spaces are a general weakness of model-free RL algorithms and the number of actions that the agent can take is usually reduced to five or less. However, if the algorithm is to be implemented in a physical device, the number of possible actions required to make the algorithm optimal and the control sequence as efficient as those obtained with classical control approaches would need to be significantly larger than those in the RL problems used for benchmarking. This would result in a significantly larger state-action space, making the training of the algorithm problematic and less efficient than Q-Learning as the algorithm would need to be m times quicker than Q-Learning. It is obvious that if m is large enough our algorithm will not be able to outperform Q-Learning. Until now the examples where the algorithm was implemented have two states and a limited number of

possible actions. If the algorithm was to be applied to control problems where there are more states or actions other solutions should be implemented. One of these solutions would be to identify the dominant states that are more influential to the performance of the algorithm so the control problem could be reduced to a two-state problem. Another alternative would be to identify the areas of the state space where the agent will be more likely present and focus the learning of the policy around these areas.

Efficiency of Training The second limitation the algorithm has to face is obtaining the correct policies in state-action pairs that are unlikely or impossible to happen because of the dynamics of the system. This limitation leads to a percentage of actions not being correctly captured in the optimal policy. This happens because the algorithm runs over all possible state-actions pairs despite some of them being unfeasible due to the dynamics. For this reason, when we look at the corners of the policies obtained we can see noise in the optimal actions that could be removed if some changes were to be applied. As the initial conditions are limited to a range of possible initial states some previous simulations could be run to obtain what are the state-action pairs that are more likely to be visited and then use the algorithm within the reachable state-action pairs to optimize the training time and get rid of the noise of wrong actions in the policy. This fact also leads usually to a residual percentage of actions that are not correctly determined. One clear example of this is in the Mountain Car problem, when the car is in a very steep position, with a velocity that points to the bottom, it does not matter which action is going to be chosen as it will always end up further down the hill. For this reason, we would recommend paying attention to this in future implementations as despite it might look like the policy is not obtained correctly, probably the controller will be able to control the agent successfully.

Assuming a Generative Model The assumption of having a generative model has been present across all experiments with Q-Learning and the Mixed Iteration algorithm. This assumption was made as all models of the environments we used were available and then we would not have to focus on determining the optimal trade-off between exploration-exploitation thus reducing the factors that could contribute to the success of the algorithm. Having the ability to sample a future state x^+ according to the probabilities that describe the dynamics of the system allows us to implement the algorithms in an easier way and focus on the behaviour and properties of the algorithm themselves. However, in most real-world applications this would not be possible and on top of implementing the algorithm, we would have to consider exploration in order to obtain the optimal policy. This remark in itself is not a limitation of the algorithm on its own but more of a reminder that in future possible applications where the algorithm is going to be applied, an ϵ-greedy update will be required instead of the updates proposed along this thesis.

7-3 Future Work

During the last section of the thesis, possible directions for future research will be appointed related to the topic discussed, and possible improvements to make the algorithm more understandable for other researchers. As the results achieved were promising according to the described metrics of success, it opens the door to other possible implementations that could be used in RL algorithms where partial information is known about the transitions between states. The four main directions where research should focus are the following ones:

- The first line of research that could be followed is using the same algorithm but instead of updating the Q-function using Q-Learning, we could use the accelerated versions of Q-Learning as Double Q-Learning or Speedy Q-Learning. If we would use the Double Q-Learning update of the Q instead of the standard Q-Learning update, the value of Q used for the update would be the one selected at that iteration at random between $Q^a(x,u)$ and $Q^b(x,u)$. By choosing this method the variance between the runs would be expected to decrease as well as the number of iterations needed for convergence. A similar better-expected performance also applies to Speedy Q-Learning. Implementing this, given the documentation and code provided, should not be challenging and despite no success being guaranteed, given the results obtained so far, there is a chance that combining Mixed Iteration with any of the two algorithms proposed would lead to improvements with respect to the basic model-free algorithms in the case of having partial information. However, notice that in both cases the per-iteration complexity would increase making the limitations stated for the Mixed Iterations algorithm more present than in the case proposed.

Model Free	**Partial Information**	**Model Based**
Q-Learning	Mixed Iterations	Value Iteration
Double Q-Learning	Double Mixed Iterations (?)	Value Iteration
Speedy Q-Learning	Speedy Mixed Iterations (?)	Value Iteration

Table 7-2: Possibilities for implementations in accelerated versions of Q-Learning

- The second proposed point that could lead to promising results is the combination of second-order algorithms with Mixed Iterations. Q-Learning is a first-order algorithm that relies on a similar behaviour to gradient descent, tracking the difference between the previous predicted value and the current one. Second-order model-free algorithms could be used, like the recently proposed Zap Q-Learning [40] in combination with its equivalent model-based algorithm, Policy Iteration. Combining these two algorithms in a similar way to the Mixed Iterations algorithm in the case of partial information could produce promising results. However, some of these second-order methods involve inverting the Hessian matrix and it largely increases the computational complexity, despite having methods to approximate it and not having to compute it analytically.

- Despite having tested the algorithm in a number of simulated environments, none of the environments were real physical devices. Implementing RL algorithms in physical devices arise challenges that were not faced in simulation environments. The first challenge is obtaining a reliable model to train the model offline. To do so, we will have to discretize the state space and be aware of the sampling time of the sensors to achieve success in control sequences. When we deal with a physical device the size of the discretization of actions and states will be usually larger than that of the RL problems where the algorithm was tested. However, testing the environment with a limited amount of actions and a smart discretization of the state space should be possible without major difficulties in the future.

- The last possible improvement that could be implemented is providing an open-source user interface. In the designed interface apart from sharing the open-source code used

in the thesis, a more user-friendly environment could be developed where the user could specify the states, actions, transition probability matrix and other necessary parameters for the algorithm. After entering the parameters specified and the number of iterations the user requires, the algorithm computes the ϵ-optimal policy and returns the evolution of the Q-cost function and the policy compared to Q-Learning and the difference in computational time between the two algorithms. By developing this interface, the user could easily evaluate if it is useful to use the algorithm or if it is better to use classical RL algorithms.

Appendix A

Proof of unique solution for the Bellman Equation

Theorem 1. *For a Markov Decision Process with a finite state-action space:*

1. *The Bellman equation has a unique solution.*
2. *The cost values obtained after the convergence of value iteration are the solution of the Bellman equation.*

The above theorem can be proved using the Banach Fixed Point theorem [20]. A fixed point equation is such that the solution of it is invariant after some given transformation. In the case of starting with any J_k and we apply the next iteration,

$$J_{k+1}(x) := \min_{u \in \mathbb{U}} \left\{ c(x,u) + \gamma \mathbb{E}_{x^+ \sim \mathbb{P}(\cdot|x,u)} \left[J_k \left(x^+ \right) \right] \right\} \quad \forall x \in \mathbb{X} \tag{A-1}$$

and we obtain a new J_{k+1}. Then we can define the Bellman Operator, $\mathcal{T}$ as the right-hand side of equation A-1. This iteration will be done repeatedly until,

$$J^* = \mathcal{T}(J^*). \tag{A-2}$$

The value of J^* is called a fixed point of the Bellman Operator as applying the operator recursively will fix the value of J^*. However, if we need to prove that $\mathcal{T}$ has the contraction property ensuring that the fixed point solution will be reached after applying a sufficient number of iterations. For an operator to be a contractor it needs to satisfy for any two cost functions J_1 and J_2 with $J_1 \neq J_2$ that after applying the Bellman Operator it will bring both them closer to the optimal J^*. The contraction property then,

$$||\mathcal{T}(J_1) - \mathcal{T}(J_2)||_\infty \leq \gamma ||J_1 - J_2||_\infty. \tag{A-3}$$

The value of $\gamma \in [0, 1)$, will be the discount factor and as it is strictly lower than 1, the norm between the two costs necessarily decreases. The Contraction Mapping Theorem states

that the Bellman Operator has a limit, the fixed point solution. Also notice that when the operator has the contraction property the solution has to be a unique fixed-point solution. The proof will be done by contradiction.

Proof. The contraction property states that there is some $\gamma \in [0, 1)$ such that: $||f(x)-f(y)|| \leq \gamma\, ||x - y|| \quad \forall x \neq y$. In the case of having two solutions x^* and y^* then,

$$||f(x^*) - f(y^*)|| = ||x^* - y^*||. \tag{A-4}$$

The equality derived contradicts the contraction property. However, if we assume there is a single x^*, then the equation holds for any $x \neq x^*$ proving that x^* is the unique fixed point solution

$$||f(x) - x^*|| = ||f(x) - f(x^*)|| < ||x - x^*||. \tag{A-5}$$

Bibliography

[1] S. R. Chowdhury and X. Zhou, "Differentially private regret minimization in episodic Markov decision processes," vol. 36, pp. 6375–6383, Jun. 2022.

[2] S. Chen and Y. Li, "An overview of robust Reinforcement Learning," *2020 IEEE International Conference on Networking, Sensing and Control (ICNSC)*, pp. 1–6, 2020.

[3] W. Yu, J. Tan, C. K. Liu, and G. Turk, "Preparing for the unknown: Learning a universal policy with online system identification," in *Robotics: Science and Systems XIII, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, July 12-16, 2017* (N. M. Amato, S. S. Srinivasa, N. Ayanian, and S. Kuindersma, eds.), 2017.

[4] J. L. Doob, "Stochastic Processes and Statistics," *Proceedings of the National Academy of Sciences*, vol. 20, no. 6, pp. 376–379, 1934.

[5] A. Markov, "Extension of the law of large numbers to quantities, depending on each other (1906). reprint.," *Journal Électronique d'Histoire des Probabilités et de la Statistique [electronic only]*, vol. 2, no. 1b, pp. Article 10, 12 p., electronic only–Article 10, 12 p., electronic only, 1906.

[6] D. R. Miller, *Markov processes*, pp. 486–490. New York, NY: Springer US, 2001.

[7] R. E. Bellman, *The Theory of Dynamic Programming*. Santa Monica, CA: RAND Corporation, 1954.

[8] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. The MIT Press, second ed., 2018.

[9] T. M. Moerland, J. Broekens, and C. M. Jonker, "Model-Based Reinforcement Learning: A survey," *CoRR*, vol. abs/2006.16712, 2020.

[10] P. Swazinna, S. Udluft, D. Hein, and T. A. Runkler, "Comparing Model-Free and Model-Based Algorithms for Offline Reinforcement Learning," *CoRR*, vol. abs/2201.05433, 2022.

[11] R. S. Sutton, *Temporal credit assignment in Reinforcement Learning*. University of Massachusetts Amherst, 1984.

[12] D. J. White, "Further Real Applications of Markov Decision Processes," *Interfaces*, vol. 18, no. 5, pp. 55–61, 1988.

[13] G. Tesauro, "Temporal Difference Learning and TD-Gammon," *J. Int. Comput. Games Assoc.*, vol. 18, p. 88, 1995.

[14] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. P. Lillicrap, K. Simonyan, and D. Hassabis, "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm," *CoRR*, vol. abs/1712.01815, 2017.

[15] O. Vinyals, I. Babuschkin, W. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. Agapiou, M. Jaderberg, and D. Silver, "Grandmaster level in StarCraft ii using multi-agent Reinforcement Learning," *Nature*, vol. 575, 11 2019.

[16] T. Kamihigashi, "Existence and Uniqueness of a Fixed Point for the Bellman Operator in Deterministic Dynamic Programming," Discussion Paper Series DP2012-05, Research Institute for Economics & Business Administration, Kobe University, Feb. 2012.

[17] E. Pashenkova, I. Rish, and R. Dechter, "Value Iteration and Policy Iteration algorithms for Markov Decision problem," 1997.

[18] B. Jang, M. Kim, G. Harerimana, and J. W. Kim, "Q-Learning Algorithms: A Comprehensive Classification and Applications," *IEEE Access*, vol. 7, pp. 133653–133667, 2019.

[19] R. Bellman, *Dynamic Programming*. Princeton, NJ, USA: Princeton University Press, 1 ed., 1957.

[20] S. Banach, "Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales," *Fundamenta Mathematicae*, vol. 3, no. 1, pp. 133–181, 1922.

[21] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3 ed., 2010.

[22] C. J. C. H. Watkins, *Learning from Delayed Rewards*. PhD thesis, King's College, Oxford, 1989.

[23] T. Jaakkola, M. I. Jordan, and S. P. Singh, "Convergence of stochastic iterative dynamic programming algorithms," in *Proceedings of the 6th International Conference on Neural Information Processing Systems*, NIPS'93, (San Francisco, CA, USA), p. 703–710, Morgan Kaufmann Publishers Inc., 1993.

[24] M. J. Wainwright, "Stochastic approximation with cone-contractive operators: Sharper l_∞-bounds for q-Learning," *CoRR*, vol. abs/1905.06265, 2019.

[25] E. Even-dar and Y. Mansour, "Learning rates for Q-Learning," in *Journal of Machine Learning Research*, pp. 1–25, 2001.

[26] J. N. Tsitsiklis, "Asynchronous stochastic approximation and Q-Learning," *Machine Learning*, vol. 16, no. 3, pp. 185–202, 1994.

[27] C. Szepesvári, "The asymptotic convergence-rate of Q-Learning," in *Proceedings of the 10th International Conference on Neural Information Processing Systems*, NIPS'97, (Cambridge, MA, USA), p. 1064–1070, MIT Press, 1997.

[28] I. Osband, D. Russo, and B. Van Roy, "(more) efficient Reinforcement Learning via posterior sampling," in *Advances in Neural Information Processing Systems* (C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, eds.), vol. 26, Curran Associates, Inc., 2013.

[29] H. Hasselt, "Double Q-Learning," in *Advances in Neural Information Processing Systems* (J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, eds.), vol. 23, Curran Associates, Inc., 2010.

[30] M. Ghavamzadeh, H. Kappen, M. Azar, and R. Munos, "Speedy Q-Learning," in *Advances in Neural Information Processing Systems* (J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, eds.), vol. 24, Curran Associates, Inc., 2011.

[31] T. Osogami, "Robustness and risk-sensitivity in Markov decision processes," in *Advances in Neural Information Processing Systems* (F. Pereira, C. Burges, L. Bottou, and K. Weinberger, eds.), vol. 25, Curran Associates, Inc., 2012.

[32] S. H. Lim, H. Xu, and S. Mannor, "Reinforcement Learning in robust Markov decision processes," in *Advances in Neural Information Processing Systems* (C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, eds.), vol. 26, Curran Associates, Inc., 2013.

[33] A. Nilim and L. Ghaoui, "Robust control of Markov decision processes with uncertain transition matrices," *Operations Research*, vol. 53, pp. 780–798, 10 2005.

[34] A. Nilim and L. Ghaoui, "Robustness in Markov decision problems with uncertain transition matrices," in *Advances in Neural Information Processing Systems* (S. Thrun, L. Saul, and B. Schölkopf, eds.), vol. 16, MIT Press, 2003.

[35] H. Xu and S. Mannor, "The robustness-performance tradeoff in Markov decision processes," in *Advances in Neural Information Processing Systems* (B. Schölkopf, J. Platt, and T. Hoffman, eds.), vol. 19, MIT Press, 2006.

[36] S. Bozinovski, "Reminder of the first paper on transfer Learning in neural networks, 1976," *Informatica*, vol. 44, 09 2020.

[37] M. E. Taylor and P. Stone, "Transfer Learning for Reinforcement Learning domains: A survey," *Journal of Machine Learning Research*, vol. 10, no. 56, pp. 1633–1685, 2009.

[38] M. K. Helwa and A. P. Schoellig, "Multi-robot transfer Learning: A dynamical system perspective," *CoRR*, vol. abs/1707.08689, 2017.

[39] N. Heess, J. J. Hunt, T. P. Lillicrap, and D. Silver, "Memory-based control with recurrent neural networks," 2015.

[40] A. M. Devraj and S. Meyn, "Zap Q-Learning," in *Advances in Neural Information Processing Systems* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, Curran Associates, Inc., 2017.