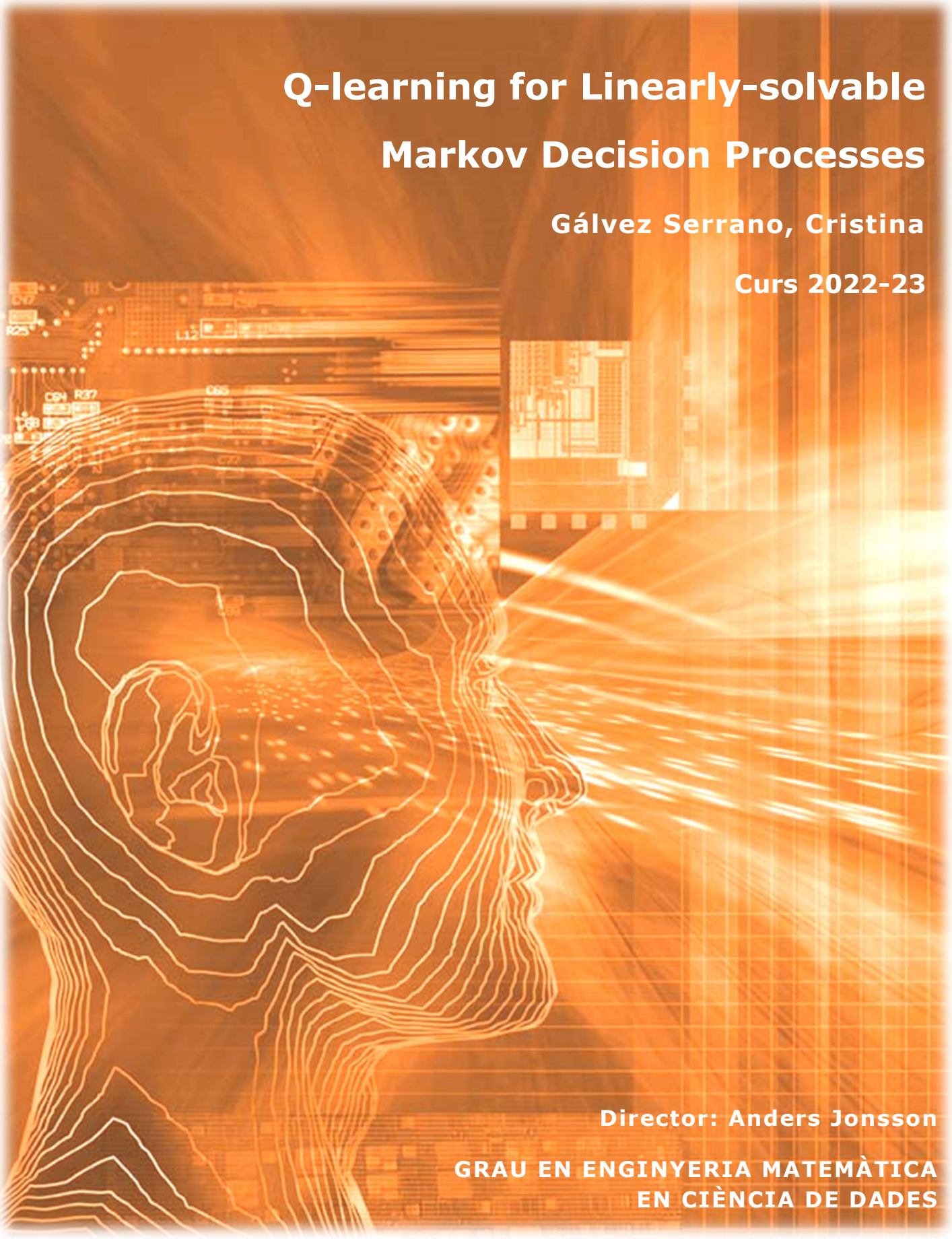

Q-learning for Linearly-solvable Markov Decision Processes

Gálvez Serrano, Cristina

Curs 2022-23

Director: Anders Jonsson

GRAU EN ENGINYERIA MATEMÀTICA EN CIÈNCIA DE DADES

upf. Universitat Pompeu Fabra Barcelona
Escola d'Enginyeria

Treball de Fi de Grau

BACHELOR DEGREE THESIS

Q-learning for Linearly-solvable Markov Decisions Processes

Cristina Gálvez Serrano

Supervised by Anders Jonsson

June 2023

upf. Universitat Pompeu Fabra Barcelona

Escola d'Enginyeria

To my family and friends

Abstract

This thesis investigates linearly-solvable Markov decision processes (LMDPs), a simplified framework derived from Markov decision processes (MDPs) in Reinforcement Learning. LMDPs aim to maximize rewards in sequential decision-making environments, providing a versatile modeling approach.

A common technique employed in MDPs is Q-learning, which approximates expected rewards for state-action pairs iteratively. In LMDPs, Z-learning assigns approximate rewards to states but requires complex modeling.

This thesis proposes the adoption of Q-learning for LMDPs, which closely resembles the structure of the MDP algorithm, computing values for each state-action pair. The newly introduced technique is evaluated in various environments, including small grid-worlds (Frozen Lake) and more complex scenarios (Sokoban). The results highlight the advantages of this method, including its fast convergence.

Resum

Aquesta tesi investiga els processos de presa de decisions de Markov resolubles linealment (LMDP), un marc simplificat derivat dels processos de presa de decisions de Markov (MDP) en Aprenentatge per Reforç. Els LMDP tenen com a objectiu maximitzar les recompenses en entorns de presa de decisions seqüencials, proporcionant un enfocament de modelització versàtil.

Una tècnica comuna emprada en els MDP és l'aprenentatge Q, que aproxima les recompenses esperades per a les parelles estat-acció de manera iterativa. En els LMDP, l'aprenentatge Z assigna recompenses aproximades als estats, però requereix una modelització complexa.

Aquesta tesi proposa l'adopció de l'aprenentatge Q per als LMDP, que s'assembla estretament a l'estructura de l'algorisme MDP, calculant valors per a cada parella estat-acció. La nova tècnica introduïda es valora en diferents entorns, incloent-hi petits mons en quadrícula (Frozen Lake) i escenaris més complexos (Sokoban). Els resultats destaquen els avantatges d'aquesta metodologia, incloent-ne la ràpida convergència.

Resumen

Esta tesis investiga los procesos de decisión de Markov resolubles linealmente (LMDP), un marco simplificado derivado de los procesos de decisión de Markov (MDP) en Aprendizaje por Refuerzo. Los LMDP tienen como objetivo maximizar las recompensas en entornos de toma de decisiones secuenciales, proporcionando un enfoque de modelado versátil.

Una técnica comúnmente empleada en los MDP es el aprendizaje Q, que aproxima las recompensas esperadas para cada par estado-acción de manera iterativa. En los LMDP, el aprendizaje Z asigna recompensas aproximadas a los estados, pero requiere una modelización compleja.

Esta tesis propone la adopción del aprendizaje Q para los LMDP, que se asemeja estrechamente a la estructura del algoritmo MDP, calculando valores para cada par estado-acción. La nueva técnica introducida se evalúa en diferentes entornos, incluyendo pequeños mundos cuadriculados (Frozen Lake) y escenarios más complejos (Sokoban). Los resultados destacan las ventajas de este método, incluyendo su rápida convergencia.

Contents

Figures

1. INTRODUCTION

1.1. Context

Reinforcement Learning (RL) is a branch of Machine Learning focused on solving sequential decision problems. These problems involve an agent that interacts with an environment, having only partial information about it. The agent observes the current state and selects actions based on a policy. After taking an action, the agent receives a numerical reward for the transition and can observe the new state.

The main objective of RL methods is to maximize the cumulative reward in order to derive efficient policies that generate optimal results for these problems. Rewards are obtained through a series of interactions called episodes. Each episode begins at an initial state and concludes at a terminal state. In some cases, episodes may be truncated if they take too long to reach a terminal state. Terminal states are defined as states from which it is impossible to transition from. Goal states are always terminal.

Sequential decision problems can be categorized as either goal-oriented or non-goal-oriented. Goal-oriented problems aim to reach a specific goal state, where the agent receives the maximum reward. Non-goal-oriented problems, on the other hand, focus on obtaining the maximum rewards before reaching a terminal state. This thesis will specifically address goal-oriented problems.

Value functions assign values to states or state-action pairs, representing the expected long-term sum of rewards. These values are then used to obtain policies, which determine the recommended actions for each state to efficiently reach the goal state.

To calculate value functions, problems are typically represented using decision control models. One commonly used model in reinforcement learning is the well-known Markov Decision Process (MDP), which provides a mathematical framework for describing how the environment behaves in response to agent actions.

Q-learning is a popular approach for obtaining value functions in MDPs. It approximates the expected total reward for each state-action pair. By computing all q-values, an efficient policy can be derived by selecting actions with the highest q-values.

Another less-known model is Linearly-solvable Markov Decision Processes (LMDPs), which simplifies the MDP model. LMDPs can represent all deterministic MDP problems while offering computational advantages.

Value functions in LMDPs are determined using a method called Z-learning, which computes z-values for each state. Implementing Z-learning can be challenging compared to other approaches because it does not use explicit actions. Furthermore, it requires the explicit probability distributions between states for its functionality, so the modelling effort is higher than with other methods.

1.2. Objectives

This thesis focuses on studying the use of the Q-learning technique for Linearly-solvable Markov Decision Processes (LMDPs). The proposed methodology extends the existing Q-learning approach and has been specifically adapted to address the characteristics of the linearly-solvable case.

The structure of the Q-learning algorithm for LMDPs resembles the Q-learning technique used in the MDP case. It computes an approximation value for each state-action pair, incorporating explicit actions into the model, which is not common in LMDPs. The q-values are iteratively computed through episodes, where rewards are received for every transition. The policy used for action selection during training resembles the approach used in Z-learning for LMDPs.

With Q-learning for LMDPs as the central focus of this study, the objectives of this document are the following:

1. Formally define Q-learning for LMDPs, presenting the underlying formulas, the structure of the algorithm, and its properties.
2. Provide examples of problems successfully solved using the Q-learning approach in LMDPs. These examples will highlight the effectiveness and applicability of the proposed methodology in different practical scenarios.
3. Conduct a comparative analysis of the performance of Q-learning for LMDPs with the two other methods mentioned in this document: Q-learning for MDPs and Z-learning for LMDPs. This analysis will assess and contrast the strengths, limitations, and practical implications of each method across different problem domains.

By achieving these objectives, this thesis aims to contribute to the existing knowledge of RL by providing a comprehensive understanding of Q-learning for LMDPs, showcasing its potential through real-world examples, and offering insights of its performance in comparison to other established techniques.

2. BACKGROUND

Sequential decision-making problems are common problems addressed in the field of Reinforcement Learning. While the specifics of these problems may vary depending on the model used, there are several key elements that are worth defining before getting into the specifics of each of the models [1].

States and State Space

In sequential decision-making problems, each instance encountered is referred to as a state. The set of all valid states form the state space. The state space represents the range of possible configurations or situations in which the agent can find itself. The size of the state space may vary depending on the complexity of the problem.

Actions and Action Space

At each state, the agent can choose from a set of actions that can lead to a transition to a new state. The collection of all possible actions in the problem is known as the action space. Similar to the state space, the size of the action space can vary depending on the problem.

Transitions

Transitions describe the process of moving from one state to another. Typically, a transition occurs by performing an action. It is important to note that not all transitions are valid, and different models may define transitions in various ways. In deterministic problems, the outcome state of a transition is predetermined and does not involve any uncertainty.

Rewards

Rewards are numerical values received by the agent for each transition, providing immediate feedback on how good a transition is for solving a problem. They are used for shaping the behavior of the agent and influencing its decision-making towards actions that maximize cumulative rewards.

Probability Distributions

Throughout the study of sequential decision-making problems, probability distributions are used to define policies, transitions, and other aspects of the models. A probability simplex over a finite set X, denoted as $\Delta(X)$, represents the set of probability distributions over X.

Mathematically, a probability distribution $p \in \mathbb{R}^X$ is defined such that $\sum p(x) = 1$ and such that $p(x) \geq 0$ for all x in X, where $p(x)$ denotes the probability of element x. In other words, the probabilities in a set X should sum up to 1 and be non-negative.

Value Functions

Value functions play a crucial role in RL. They assign values to states or state-action pairs, representing the expected long-term sum of rewards. While immediate rewards provide short-term feedback, value functions provide insights into the expected future rewards. By estimating the values of different states and actions, value functions guide the agent towards states and actions that are more likely to lead to the goal state. For example, in a grid world scenario, basic positions may receive the same immediate reward, but value functions can discern the paths that lead to higher cumulative rewards and guide the agent accordingly.

Policies

Policies determine how the agent selects actions based on the current state. Policies can be stochastic or deterministic. Stochastic policies are defined by a mapping $\pi: \mathcal{S} \rightarrow \Delta(\mathcal{A})$, where $\mathcal{S}$ represents the state space and $\mathcal{A}$ represents the action space. Deterministic policies, such as greedy policies, map each state to a single action $\pi: \mathcal{S} \rightarrow \mathcal{A}$.

Models and Methods

Models define the behavior and elements of the problem under consideration. In this section, two models will be introduced: Markov Decision Processes (MDPs) and Linearly-solvable Markov Decision Processes (LDMPs). Additionally, there are methods that are used for finding optimal policies. The ones covered in this section are Q-learning and Z-learning.

2.1. Markov Decision Processes

Markov Decision Processes (MDP) are discrete-time decision-making control models that can be used to describe sequential decision problems. An MDP is formally defined as a tuple $\{\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}\}$, where $\mathcal{S}$ represents the set of all states s in the problem and $\mathcal{A}$ represents all the actions a. The distribution $\mathcal{P}: \mathcal{S} \times \mathcal{A} \rightarrow \Delta(\mathcal{S})$ maps any state and action to new states according to a probability distribution, and the reward signal $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$, maps transitions consisting of a state and an action to a real number [1].

MDPs are based on the Markov property, which assumes that transitions are only dependent on the current state and not on the historical sequence of states. Therefore, future states are

determined solely by the current state and the chosen action, without any influence from past states [2].

In this project, it will be assumed that episodes are finite and will always end when the agent reaches a terminal state or at a specific time-step if the episode is truncated. At every time-step, the agent chooses a new action, initiating a transition that can be described as the tuple $(s_t,\ a_t, r_{t+1}, s_{t+1})$, where s_t is the current state, a_t is the action taken, r_{t+1} is the reward received for the transition, and s_{t+1} is the new state.

MDPs provide a framework for determining value functions. In the context of episodic MDPs, two types of value functions are commonly used: v-values and q-values. V-values are defined for each state, whereas q-values are defined for each pair of state and action. Both values are estimates of the expected total reward obtained by choosing actions based on a policy π.

The value function for each state in an MDP [1], given a policy π, can be formally defined as:

$$v_\pi(s) = \mathbb{E}_\pi\left[\sum_{k=t}^{T}(\gamma^{k-t} \cdot R_{k+1})\middle|\ S_t = s\right]\ with\, \gamma\ \in\ (0, 1]$$

Here, T represents the time-step at which a terminal state is reached, and γ denotes the discount factor. The discount factor determines the weighting of future rewards, giving more importance to closer time-steps compared to further ones. It is important to note that since we are assuming episodes always terminate, the discount factor γ is not essential and can be set to $\gamma = 1$ without affecting the convergence of the value function.

Another value function in MDPs is the q-value function, represented as $q_\pi(s,\ a)$, which captures the expected total reward for each pair of state and action [1]. It can be formulated as:

$$q_\pi(s, a) = \mathbb{E}_\pi\left[\sum_{k=t}^{T}(\gamma^{k-t} \cdot R_{k+1})\middle|\ S_t = s, A_t = a\right]\ with\, \gamma\ \in\ (0, 1]$$

The objective of MDPs is to find optimal policies that maximize the expected reward over time. In order to achieve this, the Bellman equations provide a set of mathematical identities that define the optimal value functions. These value functions, denoted as $v^*(s)$ and $q^*(s, a)$, are necessary for determining the optimal policies.

This alternative representation of value functions known as the Bellman optimality equations are independent of any specific policy. The Bellman optimality equations [1] are given by:

$$v^*(s) = \max_{\pi} v_\pi(s) = \max_{a} \sum_{s'} \mathcal{P}(s' | s, a) \cdot [r + \gamma \cdot v^*(s')]$$

$$q^*(s, a) = \max_{\pi} q_\pi(s, a) = \sum_{s'} \mathcal{P}(s' | s, a) \cdot \left[r + \gamma \cdot \max_{a'} q^*(s', a') \right]$$

By solving these equations, the optimal values and policies can be determined for a given MDP environment. Optimal policies can be derived by selecting the actions with the highest q-values. For example, a greedy policy based on q-values can be defined as $\pi^*(s) = \operatorname{argmax}_a q^*(s, a)$.

Solution methods for Markov Decision Processes include value iteration and policy iteration [1][2]. Value iteration is an iterative algorithm that computes the optimal value function by updating value estimates until convergence. These methods rely on having complete knowledge of the transition probabilities ($\mathcal{P}$) and reward function ($\mathcal{R}$) in order to determine the optimal policies.

In contrast, there are Temporal Difference (TD) methods like Q-learning [1], formally introduced in section 2.3. This method approximates the optimal action-value function, known as q-values, by exploring the environment and updating q-values based on observed rewards. Unlike value iteration and policy iteration, TD methods do not require explicit knowledge of $\mathcal{P}$ and $\mathcal{R}$, making them suitable for situations with incomplete information.

2.2. Linearly-solvable Markov Decision Processes

Linearly-solvable Markov Decision Processes (LMDPs) are a specific class of decision-making models that provide a framework for solving decision-making problems [3]. They are a specific case of Markov Decision Processes (MDPs) that introduce additional characteristics for efficient analysis and computation.

They were first introduced by Todorov (2006) [4], who initially formulated LMDPs without explicit actions. However, in this thesis, both cases of LMDPs will be explored, with and without explicit actions, presenting their characteristics and applications.

Not all decision-making problems can be directly modelled as linearly-solvable MDPs. Deterministic problems that can be represented as MDPs, can also be modelled as a LMDP. In these cases, taking a specific action in a given state will always result in the same subsequent state.

As these models work with deterministic problems, they can be defined without incorporating explicit actions. As transitions are deterministic, they do not depend on actions and can be just defined as the current state and the new state they transition to. The transition probability $\mathcal{P}: \mathcal{S} \rightarrow \Delta(\mathcal{S})$ represents an uncontrolled probability, which can be conveniently set as uniform. It maps a state to the set of possible new states based on a probability distribution.

The key idea for finding efficient policies in LMDPs is to update the probabilities of the policy in a way that favors transitions leading to states with higher value functions. To achieve this, the concept of Kullback-Leibler (KL) divergence is introduced [5]. This is a statistical distance that provides a measure of the difference between the distribution ($\mathcal{P}$) and the policy distribution (π). It is defined as follows:

$$KL[\pi(\cdot, s_t) \| \mathcal{P}(\cdot, s_t)] = \sum_{s'} \pi(s'|s_t) \cdot \log\left(\frac{\pi(s'|s_t)}{\mathcal{P}(s'|s_t)}\right)$$

By incorporating the KL divergence, policies are updated to maximize future cumulative reward while remaining close to the probability distribution $\mathcal{P}$. Rewards based on the policy are updated as follows, where η is a parameter to weigh the relevance of the KL-divergence term [3].

$$\mathcal{R}_\pi(s) = \mathcal{R}(s) - \frac{1}{\eta} \cdot KL[\pi(s) \parallel \mathcal{P}(s)]$$

In the following subsections, the two cases of LMDPs will be explored: explicit actions and implicit actions.

2.2.1. Implicit Actions Case

Linearly-solvable Markov Decision Processes can be defined without explicit actions. In this case, a deterministic problem can be represented with the tuple $\{\mathcal{S}, \mathcal{P}, \mathcal{R}\}$ [3][4]. The transition probability, denoted as $\mathcal{P} : \mathcal{S} \rightarrow \Delta(\mathcal{S})$, describes the uncontrolled transitions and only has non-zero values for the permitted transitions from a state s_t to s_{t+1}. In LMDPs, rewards are defined only for the current state as $\mathcal{R}: \mathcal{S} \rightarrow \mathbb{R}$. Although it is possible to define rewards over transitions, this project focuses on rewards for individual states.

The transitions are defined as (s_t, r_{t+1}, s_{t+1}), where s_t is the state from which the transition starts, s_{t+1} is the state where the transition ends and r_{t+1} is the reward received by performing it.

To compute policies in the implicit actions case, the probability distribution $\mathcal{P}$ is modified to assign greater values to better states. The reward obtained for a transition following a policy $\pi(s'|s)$ is defined using the KL-convergence. As a result, v-values [3] are expressed as:

$$v_\pi(s) = \mathbb{E}_\pi\left[\sum_{k=t}^{\mathrm{T}}\left(\mathcal{R}(\mathrm{S}_k) - \frac{1}{\eta}\cdot\log\frac{\pi(\mathrm{S}_{k+1}|\mathrm{S}_\mathrm{k})}{\mathcal{P}(\mathrm{S}_{k+1}|\mathrm{S}_\mathrm{k})}\right)\middle| S_t = s\right]$$

In LMDPs, the concept of z-values is introduced as an alternative to v-values. The z-value is obtained by exponentiating the v-value, and it is related to the v-value as follows: $z(s) = e^{\eta\cdot v(s)}$ [3]. This modification simplifies the value function, due to the presence of logarithms in the v-values.

The Bellman optimality equations for LMDPs can be expressed in terms of z-values. These equations are linear in z, which is why the name *Linearly-solvable* is used. By solving these equations, optimal policies can be determined, providing a framework for making optimal decisions.

The Bellman optimality equations for LMDPs [3] can be expressed in terms of z-values:

$$Z(s) = e^{\eta\cdot\mathcal{R}(s)}\cdot\sum_{s'}\mathcal{P}(s'|s)\cdot Z(s')$$

The optimal policy $\pi^*(s'|s)$ is calculated using the z-values. The probability of transitioning to a particular state s' from the current state s is determined by the product of the transition probability $\mathcal{P}$ and the corresponding z-value. This probability is then normalized by dividing it by the total sum of the values.

$$\pi^*(s'|s) = \frac{\mathcal{P}(s'|s)\cdot Z(s')}{\sum_{s''}\mathcal{P}(s''|s)\cdot Z(s'')}$$

2.2.2. Explicit Actions Case

In the explicit actions case of Linearly-solvable Markov Decision Process problems, the problem formulation is similar to that of MDPs. A LMDP problem is represented as a tuple $\{\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}\}$, where $\mathcal{A}$ is the set of explicit actions. Transitions have the following shape: $(s_t, a_t, r_{t+1}, s_{t+1})$, where s_t is the current state, a_t is the action taken, r_{t+1} represents the reward received, and s_{t+1} is the resulting new state.

In this case, the transition probability function $\mathcal{P}$ maps a transition to the resulting state. Since transitions are deterministic, it follows the special case of stochastic MDPs and this

probability is defined as $\mathcal{P} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$. For each pair of state and action, there is only one possible new state.

Policies π in this case are stochastic and return probability values. Policies will therefore be defined as $\pi: \mathcal{S} \rightarrow \Delta(\mathcal{A})$.

The value function in the explicit actions case is defined similarly to MDPs, but with the discount factor set to 1. Taking into account the reward definition for a given default stochastic policy π', including the KL-divergence term, the value functions can be written as:

$$v_\pi(s) = \mathbb{E}_\pi \left[\sum_{k=t}^{T} \left(\mathcal{R}(S_k) - \frac{1}{\eta} \cdot \log \frac{\pi(A_k|S_k)}{\pi'(A_k|S_k)} \right) \middle| \; S_t = s \right]$$

$$q_\pi(s,a) = \mathbb{E}_\pi \left[\sum_{k=t}^{T} \mathcal{R}(S_k) - \frac{1}{\eta} \cdot \log \frac{\pi(A_k|S_k)}{\pi'(A_k|S_k)} \middle| \; S_t = s,\; A_t = a \right]$$

In this case, the next state $S_{k+1} = \mathcal{P}(S_k, A_k)$ is completely determined by S_k and A_k since $\mathcal{P}$ is deterministic.

As mentioned in the implicit actions case, LMDPs introduce the concept of z-values, which are obtained by exponentiating the v-values: $z(s) = e^{\eta \cdot v(s)}$. The Bellman equation for z-values derived from the v-values is:

$$z(s) = \frac{1}{A} \sum_a e^{\eta \mathcal{R}(s,a)} \cdot z\big(\mathcal{P}(\mathrm{s}, \mathrm{a})\big)$$

Similarly, the q-value in LMDPs is also obtained by exponentiating the original q-value. Despite the change, the name will remain the same. The Bellman optimality equation for the LMDP q-value derived from the original q-value is:

$$q(s,a) = \frac{1}{A}\, e^{\eta \mathcal{R}(s,a)} \cdot \sum_{a'} q(\mathcal{P}(\mathrm{s}, \mathrm{a}), a')$$

In the Bellman optimality equations, a uniform reference policy is assumed. This means that the reference policy assigns equal probability $1/A$ to each action, where A represents the total number of actions in the action space.

2.3. Q-learning for MDPs

Q-learning is a well-known Temporal Difference method extensively used in Reinforcement Learning (RL) to estimate q-values within a Markov Decision Process (MDP) [2]. The concept

was originally introduced by Christopher J.C.H. Watkins (1989) [6], presenting a powerful technique for learning optimal policies without requiring explicit knowledge of the environment's dynamics. As a model-free approach, Q-learning explores and exploits actions in various states to derive the optimal policy.

This approach is particularly useful when there is incomplete knowledge about the problem, such as unknown transition probabilities ($\mathcal{P}$) or rewards ($\mathcal{R}$). It approximates q-values by observing transitions of the shape $(s_t,\ a_t, r_{t+1}, s_{t+1})$, which contain the rewards. The q-values are updated iteratively using the following formulation [1], where α_t is the learning rate and γ is the discount factor:

$$\hat{Q}(s_t, a_t) \leftarrow (1 - \alpha_t) \cdot \hat{Q}(s_t, a_t) + \alpha_t \left[r_{t+1} + \gamma \max_a \hat{Q}(s_{t+1}, a) \right]$$

The q-value approximators are initialized to 0 at the beginning of the execution and recomputed after each transition, except for terminal states where no update occurs.

An important aspect of the Q-learning algorithm is the policy used during training. It maintains a balance between exploration and exploitation. During exploration, actions are selected randomly so that all state-action pairs are seen, while during exploitation, the agent acts greedily in the current estimate of the q-values.

There are various ways to implement the action selection policy. One simple approach is to introduce a parameter $\varepsilon \in [0,\ 1]$, that determines the exploration-exploitation trade-off. Initially, ε is set close to 1 to prioritize exploration. Over time, ε gradually decreases, leading to a higher proportion of actions being selected based on exploitation rather than exploration. In this approach, a random number between 0 and 1 is generated. Depending on whether it is greater or smaller than ε, an exploration or exploitation policy is followed.

By iteratively updating q-values based on observed rewards and maximizing the value estimates, the Q-learning algorithm for MDPs can discover optimal policies.

Initially, all q-values are set to 0, and a fixed number of episodes are executed, with the environment being reset at the start of each episode. At each time step, the next action is chosen using the current policy, and the resulting reward is observed. The q-value is then updated based on the transition information. When the agent reaches a terminal state, the episode ends, and a new episode begins without any further updates to the q-values.

The Q-learning approach provides a powerful method for approximating q-values and finding optimal policies in reinforcement learning tasks. Through the iterative update process, the

action-value estimates gradually converge to the optimal action-values, reflecting the long-term expected rewards for each state-action pair.

Once the action-value function Q has been learned, an optimal policy can be derived from it. The greedy policy is represented as $\pi^*(s) = \text{argmax}_a\, Q(s,a)$. By following this policy, the agent can navigate the environment and make decisions that lead to the most favorable outcomes.

2.4. Z-learning for LMDPs

Z-learning is a specialized reinforcement learning method specifically designed for LMDPs [3]. Similar to Q-learning, Z-learning approximates z-values, which are the exponentiated form of v-values in LMDPs. In contrast to Q-learning, Z-learning requires knowledge of $\mathcal{P}: \mathcal{S} \rightarrow \Delta(\mathcal{S})$. Modeling $\mathcal{P}$ can be challenging in many problems due to complex transitions and large state spaces.

One way to compute the transition probabilities $\mathcal{P}$ is to create a matrix for each pair of states and initialize all values to 0. Then, for each state s_{t+1} that can be reached from state s_t, the corresponding entry $\mathcal{P}(s_{t+1}|s_t)$ is set to 1. If a state s_{t+1} cannot be reached from s_t, the entry is assigned a value of 0. After all entries have been assigned, the probabilities are normalized to ensure that the property $\sum_s \mathcal{P}(s|s_t) = 1$ is fulfilled.

The update rule for Z-learning involves updating the z-value estimates based on the observed transition $(s_t,\ r_{t+1},\ s_{t+1})$. The formulation for updating the z-values is given by the following expression [3]:

$$\hat{Z}(s_t) \leftarrow (1 - \alpha_t) \cdot \hat{Z}(s_t) + \alpha_t \cdot e^{\eta r_{t+1}} \sum_s P(s|s_t) \cdot \hat{Z}(s)$$

In this formulation, α represents the learning rate. This update rule allows Z-learning to update the z-values iteratively in a more efficient manner compared with MDPs.

During training, new states are selected by following a policy derived from the z-values and the probability distribution $\mathcal{P}$. This policy is denoted $\hat{\pi}$ and maps pairs of states s_t and s_{t+1} to the probability of s_{t+1} being selected while being in state s_t [3].

$$\hat{\pi}(s_{t+1}|s_t) = \frac{P(s_{t+1}|s_t) \cdot \hat{Z}(s_{t+1})}{\sum_s P(s|s_t) \cdot \hat{Z}(s)}$$

The Z-learning algorithm for LMDPs operates as follows: Initially, all z-values are initialized to 1, and a predetermined number of episodes are executed, with the environment being reset

at the beginning of each episode. During each time step, the next state is chosen based on the current policy, and the reward for the transition is observed. Using this information, the z-value of the current state is updated. Once the agent reaches a terminal state, the episode concludes, and a new episode begins.

By iteratively updating the z-values in response to the received rewards, Z-learning aims to converge towards optimal policies.

It is worth mentioning the update rule does not apply to terminal states. Instead, their z-values should be explicitly defined as $z(s_T) = e^{\eta r_T}$. If the agent lacks prior knowledge of the rewards associated with terminal states at the beginning of the execution, these values are reassigned every time the agent reaches a terminal state.

3. CONTRIBUTION

3.1. Q-learning for LMDPs

In this section, I introduce a new Temporal Difference method called Q-Learning for Linearly-solvable Markov Decision Processes (LMDPs). As other similar methodologies, its goal is to find optimal policies by approximating value functions to their optimal values. Specifically, Q-Learning for LMDPs focuses on optimizing the value function known as LMDP q-values, which gives the name of Q-learning to this method.

Q-learning for LMDP is an adaptation of the original Q-learning algorithm designed to conserve the convergence properties of LMDPs in a simplified manner compared to existing algorithms for this class. While explicit actions may seem unnecessary for LMDPs, this method does use them. Explicit actions offer several implementation advantages, especially in frameworks that use action transitions.

Similar to the original Q-Learning, this approach computes q-values for each state-action pair. The q-values are initialized to 1 and updated through a series of episodes using an update rule. It should be reminded that the q-values computed in this approach are the exponentiated versions of the traditional q-values for MDPs.

As with other Temporal Difference methods, Q-Learning for LMDP involves using a policy to select actions. This policy is updated based on new q-value approximations, gradually converging towards the optimal policy. The policy, denoted as $\pi(a|s)$, is derived from the q-values by normalizing all q-values for a given state and using the resulting values as the probabilities of selecting each action. This can be expressed as:

$$\pi(a|s) = \frac{\hat{Q}(s,a)}{\sum_{a'} \hat{Q}(s,a')}$$

Unlike the original Q-learning, this adaptation does not require explicit control of the exploration-exploitation ratio. Instead, this ratio is indirectly influenced by the policy. Initially, actions are selected in an exploratory way. All actions have an equal probability of being selected because the q-values are initialized with the same value. As the learning progresses, actions with higher q-values become more likely to be chosen, promoting exploitation.

Q-values are initially set to $\hat{Q}(s,a) = 1$, for all states s and actions a. Then, these q-values are updated following the update rule based on the observed transition $(s_t,\ a_t, r_{t+1}, s_{t+1})$.

This update rule assumes a uniform default policy π', that selects each action with a probability $1/A$.

$$\hat{Q}(s_t, a_t) \leftarrow (1-\alpha) \cdot \hat{Q}(s_t, a_t) + \alpha \frac{1}{A}\, e^{\eta r_{t+1}} \sum_{a'} \hat{Q}(s_{t+1}, a')$$

In this formulation, α represents the learning rate and η denotes the reward factor. This update rule incorporates the exponentiated rewards, which is one characteristic distinction of LMDPs compared to the traditional Q-learning update rule. The exponent term is multiplied by the average of q-values at the new state s_{t+1}, represented by the summation of q-values over the denominator A, indicating the number of actions in the action space.

During execution, the policy is updated based on the q-values. This policy, denoted $\pi: \mathcal{S} \rightarrow \Delta(\mathcal{A})$, is updated until it converges to the optimal policy. Through the iterative updates of q-values and the derived policies, $\pi^*(s)$ is efficiently computed.

3.2. Algorithm

In this section, I am presenting the algorithm and providing an explanation of the Q-learning process specific to LMDPs. This algorithm shares a similar structure with its predecessors, Q-learning for MDPs and Z-learning for LMDPs.

To begin, we need to create a data structure to store the q-values. A convenient approach is to use a matrix that matches the dimensions of the state space and action space.

Next, the agent will proceed with running episodes. During each episode, the q-values will be updated for each transition. At each step, the agent selects an action based on the policy and receives a numerical reward. This reward is then used to update the corresponding q-value following the defined update rule. If a transition leads to a terminal state, the episode terminates, and a new one begins.

The policy utilized in this learning process relies on the q-values and evolves as the training progresses. It is determined by normalizing the estimated q-values at a given state and using these values as the probabilities for selecting each action.

3.3. Characteristics and Applicability

Q-learning for LMDPs is a method specifically designed for solving deterministic problems. It works best with discrete state and action spaces, and it would need to be adapted to the continuous case. In those instances, we can discretize the spaces or implement q-values as a

function with some parameters to be approximated. However, this thesis focuses on discrete problems and does not cover continuous spaces.

One great advantage of Q-learning for LMDPs is its flexibility. It can handle various problems without needing to know the exact transition probabilities or rewards in advance. The method learns from experience, gradually estimating q-values and developing efficient policies. By exploring the environment and observing rewards, the algorithm refines its q-value approximations, leading the agent to make better decisions over time. Hence, it is a method that can adapt to very different problem domains, even when the inner workings are unknown.

In Q-learning for LMDPs, less modeling effort is required compared to Z-learning, assuming access to a simulator. By having information about a simulator, the explicit modeling of transition probabilities is avoided. This simplifies the implementation process, allowing a focus on using observed rewards to approximate optimal actions.

Another important aspect is that Q-learning for LMDPs is believed to converge for all discrete deterministic problems with well-defined rewards. Proving convergence is, nevertheless, outside the scope of this thesis.

In summary, Q-learning for LMDPs is an effective approach for approximating q-values and finding optimal policies in discrete deterministic problems. While it is most suitable for discrete scenarios and may not handle continuous spaces as well, it offers the ability to learn from experience and adapt to unknown probabilities and rewards. By converging towards optimal policies, it proves to be a reliable framework for solving a wide range of discrete deterministic problems.

4. RESULTS

4.1. Execution of Q-learning for LMDPs

As part of this research, Q-learning for LMDPs has been implemented and various experiments have been conducted to evaluate its performance. The algorithms used were developed following the provided pseudocode and implemented using the Python programming language. This allowed for testing its usage, measuring its performance, and comparing it with the other two methods: Q-learning for MDPs and Z-learning for LDMPs.

In the upcoming sections, more detailed explanations will be provided about the framework employed and the specific environments and problems that were addressed in this study.

The code used for conducting the experiments is publicly available in a GitHub repository[1]. This repository contains the code implementations used to replicate and verify the results presented in this study, facilitating public access for further investigation and analysis.

4.1.1. Framework

The framework used in these experiments consists of Python programming language and Jupyter notebooks. Python offers a wide range of libraries and tools suitable for reinforcement learning tasks [7]. On the other hand, Jupyter notebooks provide an interactive coding environment for the Python programming language, facilitating the development process [8].

To ensure consistency across the learning methods, they were coded as Python functions that closely followed the algorithms described earlier. For instance, the policies are coded as separated functions and value-functions are implemented as matrices. This allowed to easily compare and analyze the performance of different algorithms.

For the agent-environment interaction, the gym library was employed. Gym is a powerful Python library specifically designed for reinforcement learning. It provides pre-defined environments, including classic control problems, which proved useful for the experiments conducted [9]. However, custom environments based on the gym class structure were created to focus on discrete state spaces.

[1] Q-learning for LMDPs, GitHub Repository *https://github.com/CrisGalv/Q-learning-for-LDMPs*

With this framework and environment setup, it was possible to conduct comprehensive experiments and evaluate the effectiveness of Q-learning for LMDPs in solving various problem domains.

4.1.2. Environments

The environments used for training the policies were custom-built problems inspired by well-known scenarios. These environments are based on grid worlds, which are a straightforward approach to defining discrete space environments that can be adjusted in complexity.

Frozen Lake

Frozen Lake is a simple yet engaging toy problem. It involves an agent placed in a marked position and its objective is to reach a target position without falling into specific cells that act as holes. Both the hole cells and the target cell are considered terminal states.

The state space is defined by the position of the agent on the grid, while the action space consists of the four cardinal directions. For this implementation, a 6x6 grid with 6 holes was employed.

The figure below illustrates the initial state of the implemented Frozen Lake. The agent, represented by the black circle, must navigate through the grid cells to reach the target (yellow cell) while avoiding the holes (blue cells).

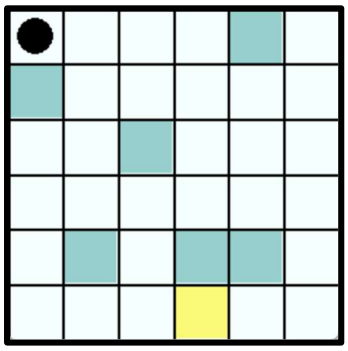

Figure 1: Frozen Lake environment

Sokoban

Sokoban is a popular game in which an agent needs to strategically move boxes to their designated target positions. The agent can only push the boxes, and there are various terminal states, such as when a box is trapped in a corner that is not its target position. Despite its simple nature, Sokoban can present considerable challenges [10].

In this implementation, three different levels were defined, with the only variation being the target positions for the boxes. The state space in Sokoban is more complex compared to Frozen Lake, as it stores the positions of the two boxes and the agent.

The three implemented levels are depicted in the following figures. The agent is represented by a black circle, and the boxes are shown as brown squares that need to be moved to the target positions indicated by yellow cells. Additionally, there are walls represented in dark blue, which act as obstacles that cannot be crossed. The difficulty of each level is determined by the optimal number of steps required to reach the goal state.

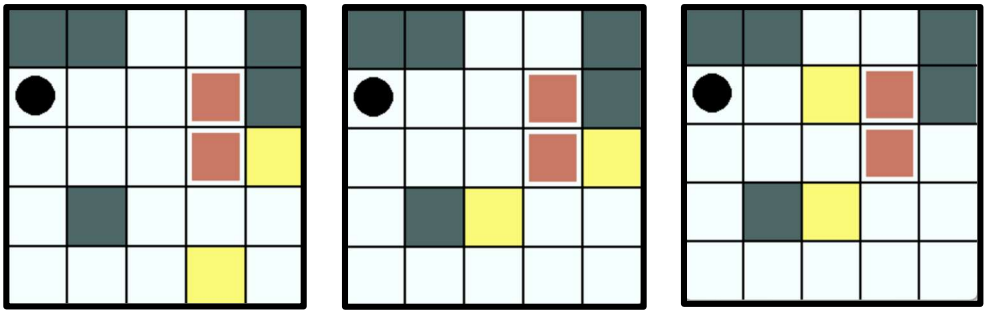

Figure 2: Levels of Sokoban environment

Both environments used in this study have been developed by extending the environment class of the gym library. All fundamental methods have been overridden to behave accordingly with the specific problem definitions.

4.2. Comparison over Training

To evaluate the performance of Q-learning for LMDPs, it has been compared with two other methods: Q-learning for MDPs and Z-learning for LMDPs. This comparative analysis is based on various metrics obtained throughout the learning process, such as the total reward, the time needed or the convergence ratio.

Multiple experiments were conducted in this study, training the different models using the covered methods. All experiments utilized a fixed learning rate of $\alpha_t = 0.1$ for all time steps, ensuring consistency in the learning process.

4.2.1. Total Reward

One approach to comparing the methods is by considering the total reward obtained by the agent in each episode during the training process. This measure, referred to as the 'score' of the episode, provides a way to assess and compare the performance of the methods. By analyzing the scores achieved by the agent and how they evolve during the training process, we can gain insights into the effectiveness of each method.

It is important to note that the policies used to select actions (or states in the case of Z-learning) change over time and can impact the scores. This effect is particularly noticeable in Q-learning for MDPs. Due to the linear change of epsilon in my implementation, the method may already have good approximations of q-values midway through execution but is still forced to explore, resulting in lower scores.

The plot below illustrates the scores obtained by each method over time. Z-learning obtains the highest scores and achieves them more rapidly, showcasing the computational advantages of LMDPs compared to generic MDPs. Q-learning for LMDPs also performs well and exhibits faster convergence compared to the original Q-learning for MDPs, which shows slower progress in obtaining desirable results.

It must be considered that the following resulting plot lines have been smoothed by taking the average by chunks and a moving average filter.

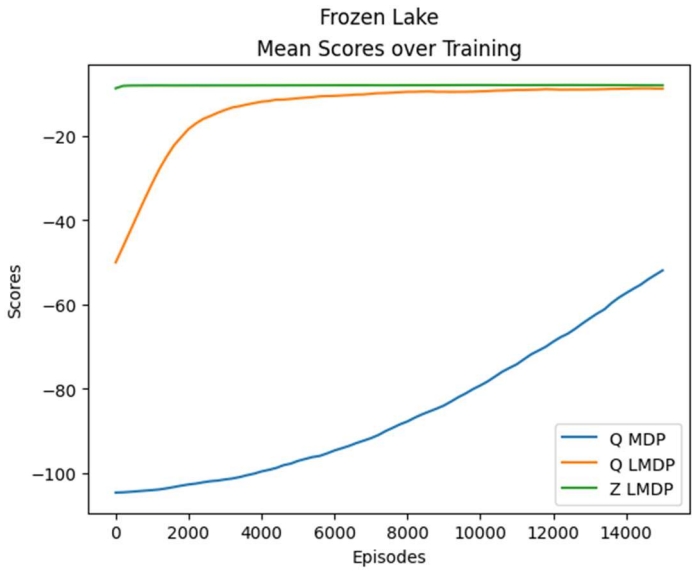

Figure 3: Scores comparison for Frozen Lake with $\alpha = 0.1$

For simple problems, for example Frozen Lake, that has a small state space and the solution is very easy to find, the scores obtained with the Q-learning for LMDPs approach very quickly resemble the ones from the Z-learning. This phenomenon can also be seen for Sokoban level 2. Although the state space is greater, the optimal solution is still relatively easy to find.

On the other hand, the results obtained for Sokoban level 3 are completely different. The scores for Q-learning for LMDPs are only slightly better than the ones obtained with the original Q-learning, as it is depicted in *Figure 5*.

This level is particularly challenging compared to the others, and it is uncommon for the agent to find the solution through random exploration. The optimal solution can only be achieved by carefully following a specific, relatively lengthy path.

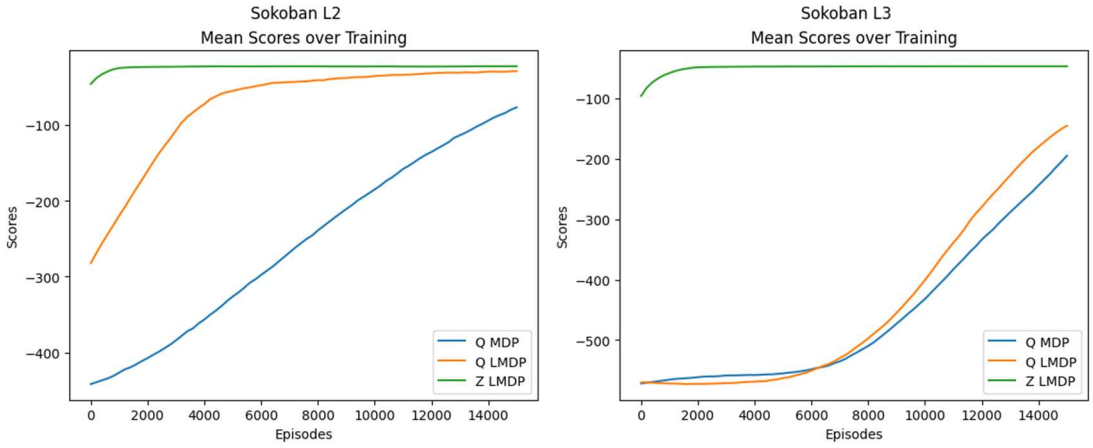

Figure 4: Scores comparison for Sokoban levels 2 with $\alpha = 0.1$

Figure 5: Scores comparison for Sokoban levels 3 with $\alpha = 0.1$

Therefore, it can be concluded that the scores over training of Q-learning for LMDPs are better than the ones obtained with the original Q-learning method. Nevertheless, they depend on the problem definition and how likely it is to find a solution by taking random steps. The more likely it is to find a solution, the better scores Q-learning for LMDPs will obtain.

4.2.2. Execution Time

Another important metric to consider is the execution time. So far, the previously mentioned metric of scores over training episodes does not account for time. Different methods involve different computations, resulting in varying execution times.

Moreover, the complexity of the problems being solved significantly impacts the execution timings. For simpler problems with small state spaces, the differences between methods may not be very noticeable. However, for more complex problems with larger state spaces, longer computations can considerably extend the overall execution time.

Since the number of steps can vary across episodes, the timing has been measured in milliseconds per step. The boxplots presented in *Figures 6 and 7* illustrate the average time needed to complete a time step for each of the different methods in various problem domains.

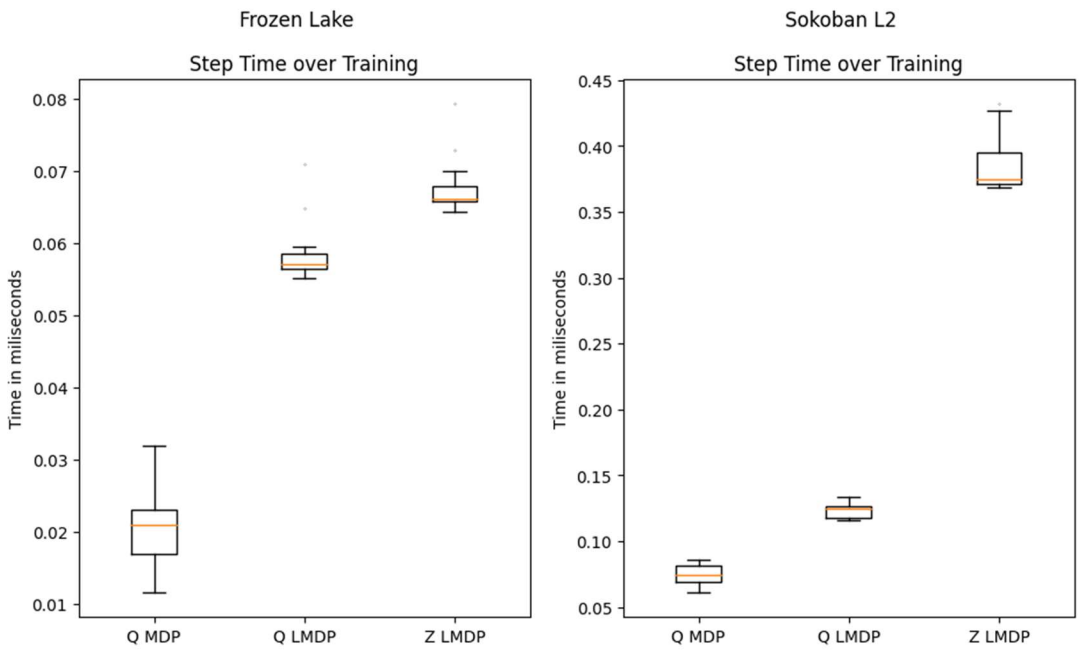

Figure 6: Execution time per step in Frozen Lake

Figure 7: Execution time per step in Sokoban level 2

It is worth noting that although these problems have relatively small state spaces, as the state space increases in size, computations, especially in Z-learning, can take significantly more time. As it can be observed in the plots, the difference between the two LMDP methods is much larger for the Sokoban environment, which is more complex than Frozen Lake.

The plot presented below illustrates the time distribution of the main operations in the different algorithms. The timing was conducted specifically on Sokoban level 2, and only sections with significant time consumption are included in the plot.

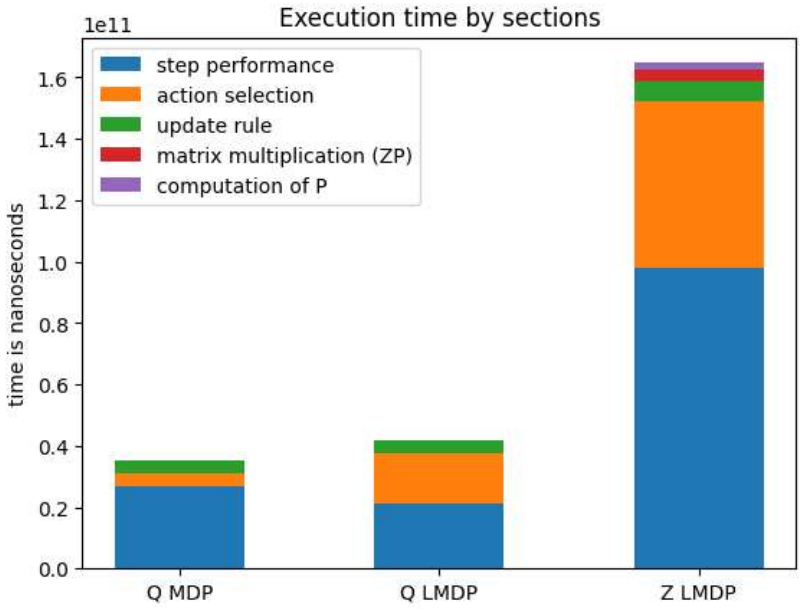

Figure 8: Execution time per sections in Sokoban level 2

As depicted in *Figure 8*, the most significant amount of time is allocated to the step performance, which involves computing the necessary changes in the problem for transitioning. In the case of Z-learning, which uses the implicit actions case of LMDPs, more time is required for these steps.

Similarly, the absence of explicit actions in Z-learning leads to slower action selection. Choosing between a small set of actions versus selecting from an array containing all possible states can result in significant differences in computational efficiency. The difference in size between the action space and the state space can have a notable impact on performance. Additionally, working with state-to-state mappings, considering this size disparity, tends to be more computationally expensive.

Regarding Z-learning, it is apparent that the computation of the uncontrolled transition probability matrix, P, and the subsequent matrix multiplication, represented as $\sum_s P(s|s_t) \cdot \hat{Z}(s)$, introduce additional overhead that is not present in the other two methods. This contributes to the overall performance disparity observed in the experiments.

4.2.3. Convergence

Convergence is another metric that can be obtained from the learning process and measures the proximity to the optimal solution at each time-step. It measures the error between different instances of the value function estimators.

In this study, the convergence ratio was assessed at regular intervals of every 10 iterations. The value-function approximations were stored, and a comparison was made between the current matrix and the previously saved instance. This comparison was based on the mean squared error, which involved calculating the squared difference between the matrices and then computing the mean of all the differences.

Notably, there is a noticeable difference in convergence between methods designed for MDPs and those for the Linearly-solvable case (LMDPs). Both Q-learning and Z-learning approaches for LMDPs show significantly faster convergence towards their optimal values. This trend is clearly shown in the following plots.

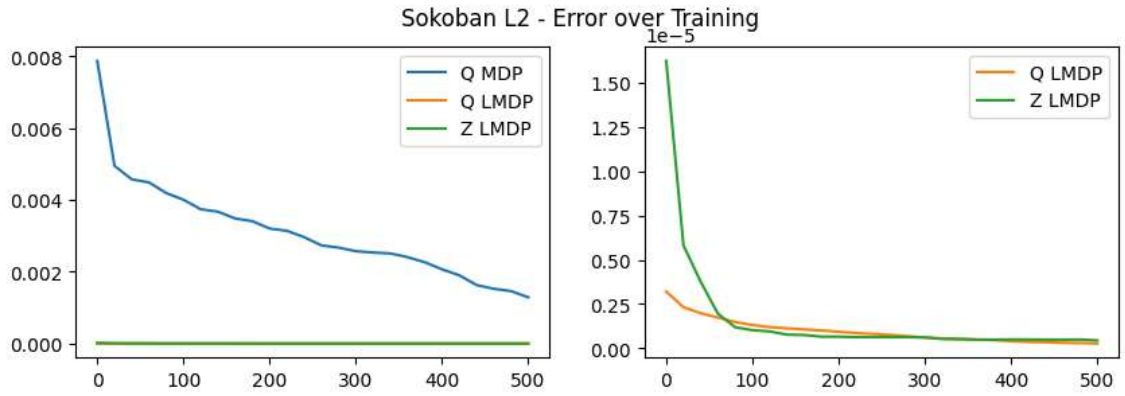

Figure 9: Convergence rate for Sokoban level 2

The comparison of convergence ratios reveals that the original Q-learning method is slower in convergence compared to the other two approaches. This can be observed in the first plot. In contrast, the second comparison plot demonstrates that both methods designed for LMDPs, Q-learning and Z-learning, show remarkably similar and rapid convergence.

4.3. Comparison of Resulting Policies

In the previous sections, the comparison between methods covered focused exclusively on metrics obtained during training. However, it is also important to consider the resulting policies to assess the effectiveness of these learning methods.

When trained for a sufficient amount of time with appropriate parameters, all methods have the potential to reach the optimal policy. However, when using Q-learning for MDPs with an explicit exploration-exploitation ratio, there is a risk of shifting too quickly towards exploiting known information instead of exploring new possibilities. This can lead to the derivation of suboptimal policies.

In contrast, the other two methods, Q-learning for LMDPs and Z-learning, have self-regulating policies during training. These methods have built-in mechanisms to balance exploration and exploitation without the need for manual control. As a result, they are less likely to prematurely favor exploitation over exploration and are more likely to converge to optimal policies.

In terms of resulting policies, Q-learning and Z-learning for LMDPs typically obtain the same outcome: the optimal policy. On the other hand, the original Q-learning method may fail to converge to the optimal policy within the same number of episodes as the other two methods.

5. CONCLUSION

Q-learning for Linearly-solvable Markov Decision Processes (LMDPs) introduces a fresh approach that stands out for its fast convergence and easy implementation. This method offers unique benefits and addresses specific challenges encountered in the world of temporal-difference methods.

A notable feature of Q-learning for LMDPs is its capability to handle problems involving explicit actions, which is uncommon in the LMDP framework. Unlike traditional LMDP formulations that primarily focus on state transitions, Q-learning expands the scope by incorporating explicit representations of actions. By explicitly modeling actions and their associated q-values, this method provides a comprehensive framework for learning and deriving optimal policies in LMDPs.

Empirical evaluations of Q-learning for LMDPs have demonstrated its effectiveness and superiority in various domains compared to the standard MDP case. Notably, this approach has shown impressive results in terms of performance metrics such as scores, completion time, and convergence.

One of the key advantages of Q-learning for LMDPs is its minimal modeling overhead and effort required for implementation. Unlike other reinforcement learning approaches that require extensive modeling, Q-learning for LMDPs offers a straightforward implementation process. By reducing the complexity associated with modeling, this method can be easily adapted to multiple decision-making scenarios.

In summary, Q-learning for Linearly-solvable Markov Decision Processes presents a valuable approach characterized by its quick convergence and user-friendly implementation. Its ability to handle explicit actions and its strong empirical performance make it an attractive option for solving complex decision-making problems. By minimizing the modeling complexity and providing reliable results, this method holds promise for diverse applications in the field of intelligent systems.

Bibliography

[1] Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (2nd ed.). Bradford Books.

[2] Russell, S., & Norvig, P. (2021). *Artificial intelligence: A modern approach* (4th ed.). Pearson.

[3] Jonsson, A., & Gómez, V. (2016). Hierarchical linearly-solvable Markov decision problems. In *arXiv [cs.AI]*. http://arxiv.org/abs/1603.03267

[4] Todorov, E. (2006). *Linearly-solvable Markov decision problems* (Vol. 19). MIT Press.

[5] Joyce, J. M. (2011). Kullback-Leibler Divergence. In *International Encyclopedia of Statistical Science* (pp. 720–722). Springer Berlin Heidelberg.

[6] Watkins, C. J. C. H., & Dayan, P. (1992). Q-learning. *Machine Learning*, *8*(3–4), 279–292. https://doi.org/10.1007/bf00992698

[7] *Python*. (n.d.). Python.org. Retrieved May 29, 2023, from https://www.python.org/

[8] *Project Jupyter*. (n.d.). Jupyter.org. Retrieved May 25, 2023, from https://jupyter.org/

[9] *Gymnasium documentation*. (n.d.). Farama.org. Retrieved May 25, 2023, from https://gymnasium.farama.org/

[10] Junghanns, A., & Schaeffer, J. (2001). Sokoban: Enhancing general single-agent search methods using domain knowledge. *Artificial Intelligence*, *129* (1–2), 219–251. https://doi.org/10.1016/s0004-3702(01)00109-6